# Requirements Documents Evolution and Synchronization with Activities in the Refined Requirements Generation Model

Ulziidelger Magsarjav

Thesis submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
Computer Science and Applications

Approved:

_____

James D. Arthur, Chair

_____ _____

Osman Balci Roger W. Ehrich

September, 2004

Blacksburg, Virginia

Keywords:
Requirements Generation, Software Engineering,
Requirements Evolution, Requirements Documents

# Requirements Documents Evolution and Synchronization with Activities in the Refined Requirements Generation Model

Ulziidelger Magsarjav

## ABSTRACT

Over the past few years the real importance of requirements engineering has surfaced; hence, much research is now being directed towards generating quality requirements. However, the existing requirements generation models do not sufficiently stress the importance of identifying intermediate requirements documents. In addition, the models rarely specify how those documents support the objectives of the related activities. Moreover, the current models fail to depict how requirements are transformed, in terms of content and format, as we transition through the requirements engineering process.

To address these concerns, we propose a comprehensive requirements generation model consisting of two main parts – (1) a refined set of activities (of the model) with explicitly enunciated objectives, and (2) a detailed characterization of requirements documents generated throughout the requirements engineering process.

The proposed model refines the Requirements Generations Model (RGM) into detailed activities to reflect an appropriate level of abstraction, so that we can more accurately represent the intermediate development of the requirements documents. Furthermore, the objectives of the activities are identified, and subsequently, synchronized with the content and format of the documents produced by each activity. The evolution of the requirements is described, in terms of content and format, as the requirements documents pass through the successive activities of the requirements engineering process.

# ACKNOWLEDGMENTS

I am grateful to my advisor Dr. James Arthur for his support and guidance. His direction, motivation, and inspiration lead to the accomplishment of my thesis. Furthermore, I am thankful to Dr. Osman Balci and Dr. Roger Ehrich for serving on my thesis committee.

I am grateful to my family for supporting me throughout my life. In addition, I would like to thank my friends for their encouragement and positive influence.

Last but not least, I am thankful to the computer science department – faculty, staff, and fellow students – for being an important part of my overall learning experience.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1. INTRODUCTION

## 1. Introduction

The requirements generation process consists of various activities, starting with the elicitation of user needs and extending to the generation the software requirements specification (SRS). This thesis presents the decomposition and refinement of the Requirements Generation Model (RGM) [Arthur 1999]. The refinement is focused on the recording and usage of requirements documents as they evolve throughout the requirements engineering process. In addition, this research also provides detailed information about the intermediate and final representations of requirements and presents the templates for those documents.

This chapter motivates the need for continuous research in the requirements engineering discipline and the reasons for supporting the requirements generation process with structured requirements documents.

## 1.1. Motivation

The following section describes the necessity for this research in the requirements engineering field. First, we discuss the importance and roles of requirements engineering and how it was recognized as a vital part in the software development process. We then examine the problems faced in requirement engineering and present current approaches to addressing them. We also outline what these current approaches lack, since their shortcomings are the motivating factors for our research.

### *1.1.1. Software Development Life Cycle*

The earliest approaches to software development were *adhoc* or programming centered, which considered development of software primarily as a programming effort. Such approaches resulted in the projects being late, costly, and unreliable. Moreover, the

maintenance on the delivered product was significant – adding additional expenses to the project.

In the 1960s the introduction of third-generation computer hardware enabled large software systems to be built [Sommerville 1996]. The problems encountered in building large software systems were not simply scaled up versions of the problems of developing small computer programs. As a result, the cost of building large software systems increased dramatically [Boehm 1976]. One answer to the increased cost of software development was "software engineering", first introduced in the late 1960s at a conference held to discuss the "software crisis" [Sommerville 1996]. Software engineering is formally defined as [IEEE 1987]:

> *…a systematic approach to the development, operation, maintenance, and retirement of software.*

The objective of software engineering is to identify methods and procedures for the development of software that can scale up for large projects, and be used to consistently produce high-quality software at low cost [Jalote 1999]. Thus, the key objectives are consistency, low cost, high quality, and scalability.

The initial model of the Software Development Life Cycle (SDLC) was the waterfall model produced by [Royce 1970]. Since then, several approaches such as prototyping and spiral have improved upon the waterfall model. This has resulted in more robust methodologies for the development of software. In spite of the differences in the approaches to the SDLC, most of the models encompass the same "macro" development phases – requirements analysis, software design, coding and unit testing, system testing, and maintenance [Figure 1.1].

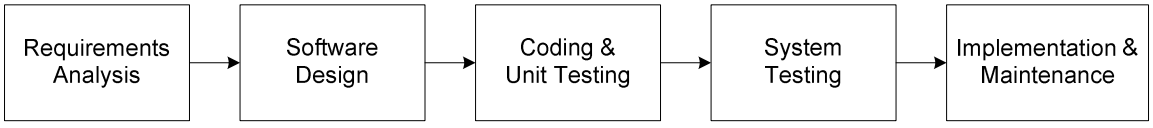



*Figure 1.1 – Main phases in the SDLC*

Despite the number of approaches proposed, software development was still facing the earlier problems of cost and schedule delays because the phases of the models lacked the

necessary support in terms of guidelines, tools and methods. As a result, emphasis was placed on improving the individual phases. In spite of being "first" in the sequence of the SDLC phases, requirements analysis has been the last to be re-examined and refined by the software engineering community [Sidky 2002]. The refinement of the phases proposed by Royce took place in a backward direction starting with testing. Several tools, methods, and guidelines were developed for the testing phase, such as black box and white box testing techniques. Coding was also enhanced with the integrated development environment (IDE), coding guidelines and error detectors, and a number of other tools. The design phase was improved with notations such as UML and supporting tools.

However, in spite of these advances in the various SDLC phases, the product often failed to meet the user's intent. The main reason was the lack of a requirements generation framework and suitable guidelines. It is only in recent years that the importance of requirements has been recognized and efforts are being made to overcome the problems in the requirements analysis phase. Moreover, since the name requirements analysis was inappropriate to cover all requirements activities, it was changed to requirements engineering, which includes the activities – elicitation, analysis, specification, verification and management. The following section highlights the importance of requirement engineering in the SDLC.

### *1.1.2. Importance of the Requirements Engineering*

Success in software development is measured by the quality of the product delivered to the customer. Requirements engineering is a critical phase for the success of a project since this phase ensures that the software system reflects the customer needs. The requirements engineering phase focuses on deciding precisely what to build and this is the most difficult part of building a software system [Brooks 1987].

A good set of requirements cannot be obtained by an *adhoc* process; instead, they need to be engineered through a systematic and well-defined process [Bell 1976]. Therefore, as mentioned in the previous section, the requirements analysis phase in the SDLC has evolved into the requirements engineering, which is defined as "a systematic process of developing requirements through an iterative co-operative process of: analyzing the

problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained" [Macaulay 1996].

The objective of the requirements engineering phase is to obtain a complete and clear software requirements specification (SRS), which captures the user's intent. However, if the requirements engineering process fails to generate quality requirements, the potential impact is substantial. Errors committed during the requirements phase often remain latent and are not detected until well after the stage in which they are made. The later in the development life cycle that a software error is detected, the more expensive it is to repair. In addition to increasing cost and schedule, errors in the requirements often lead to the development of a wrong product, which does not meet the user expectations.

A number of studies have been conducted to quantify the impact of requirements errors in the software development process. For example, three companies (GTE, TRW, and IBM) performed independent studies to measure the expenses of repairing errors in different phases of the SDLC. Even though these studies were conducted independently, they all reached roughly the same results depicted in Figure 1.2 [Davis 1993]. The figure illustrates that the cost to fix requirements errors increase rapidly in the later development phases. The relative costs are calculated by making the assumption that a unit cost of detecting and repairing a software error in the coding phase is assigned a value of one. The surveys show a very important fact that detecting an error in the requirements phase could save a software repair cost 200 times than if it is detected in the maintenance phase.

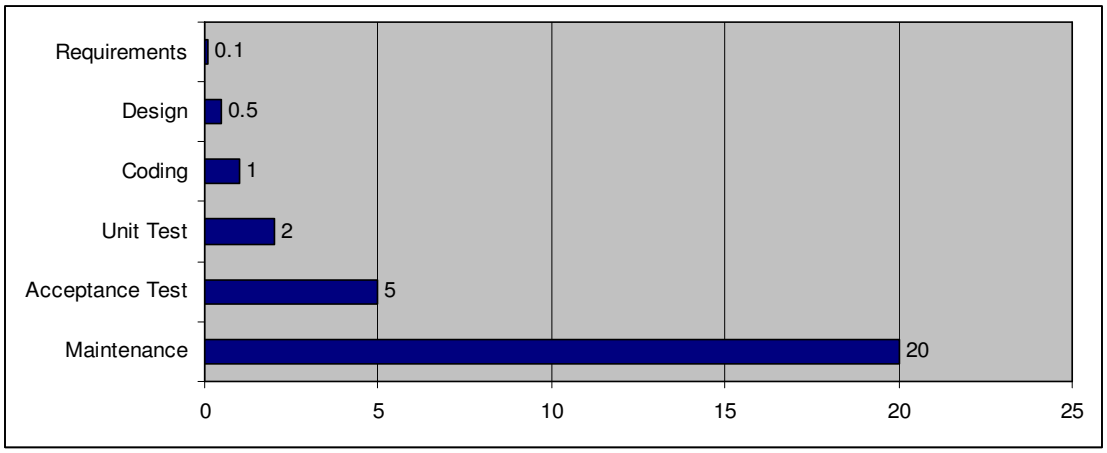



*Figure 1.2 – Relative cost to repair a defect at different software life cycle phases*

In recent years, an increased awareness of requirements engineering in the software industry is becoming more apparent because of certain alarming facts. For instance, the CHAOS report by the Standish Group reveals the significance of the requirements engineering in the software development. According to the CHAOS report, three top challenging factors in the software development are lack of user input, incomplete requirements specifications and changing requirements [Standish 1995]. Furthermore, the report emphasizes the importance of involving users in the requirements generation process, and clearly stating requirements as part of top project success factors.

Another large-scale survey conducted by the ESPITI (European Software Process Improvement Training Initiative) indicates that two largest problems in the software industry are (1) requirements specifications, and (2) managing customer requirements [ESPITI 1995]. Detailed results are indicated in Figure 1.3.

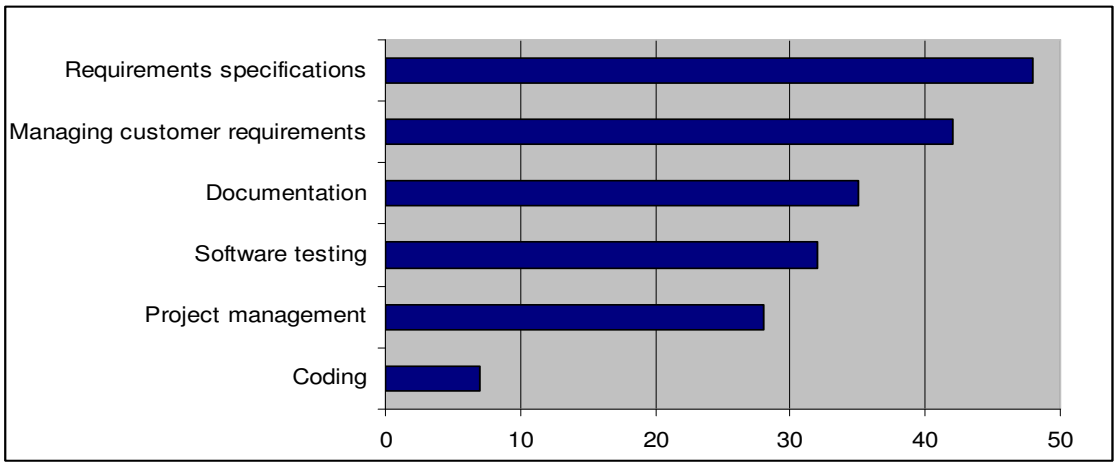



*Figure 1.3 – Major problems in software development*

Over the past few years, the software industry has realized the importance of requirements engineering, and a number of research efforts are being directed towards generating quality requirements.

### *1.1.3. Current developments in the Requirements Engineering*

The requirements engineering process is divided into five major activities, and current research is focused on improving each individual activity as well as integration among

them. A brief description of the requirement engineering activities adapted from [Thayer 1997] are listed below:

- Requirements elicitation – The customers and developers of a software system discover, review, articulate, and understand the user needs and constraints on the software and the development activity.
- Requirements analysis – The process of analyzing the customer and user needs to arrive at a definition of software requirements. The customer assesses the acceptable level of risks regarding completeness, correctness, and technical and cost feasibility.
- Requirements specification – The requirements elicited and analyzed in the preceding activities are documented in the form of a formal document, often referred to as the Software Requirements Specification (SRS).
- Requirements verification – The process of ensuring that the requirements elicited and documented in the SRS comply with the system requirements and customer needs. The requirements are also verified for conformance to document standards and adequate basis for the architectural (preliminary) design phase.
- Requirements management – It is "a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system" [Leffingwell 2000]. The management activity pervades the entire requirements engineering process ensuring the planning and control of the requirements elicitation, analysis, specification, and verification.

Significant research is being conducted in the requirements engineering field; however, the industry still fails to produce quality requirements. Errors attributed to the SRS are typically incorrect facts, omissions, inconsistencies, and ambiguities (Figure 1.4) [Basili 1981].

The reasons for incorrect and incomplete requirements are mainly due to the intrinsic problems in the requirements generation process. For example, communication with the customer/user is vital in the requirements generation process because the requirements engineer has a responsibility to bridge the user application domain and the solution

domain. Communicating in natural language can easily lead to ambiguity and/or misunderstanding. Often times the requirements engineer does not have the adequate knowledge about the problem domain; therefore, understanding user domain terminologies and ensuring that the real user needs are reflected in the software requirements is challenging. Furthermore, a lack of user participation has been a major problem [Standish 1995] due to both users and developers having insufficient understanding about the importance of user input in generating quality requirements, which in turn drives the success of the software system.

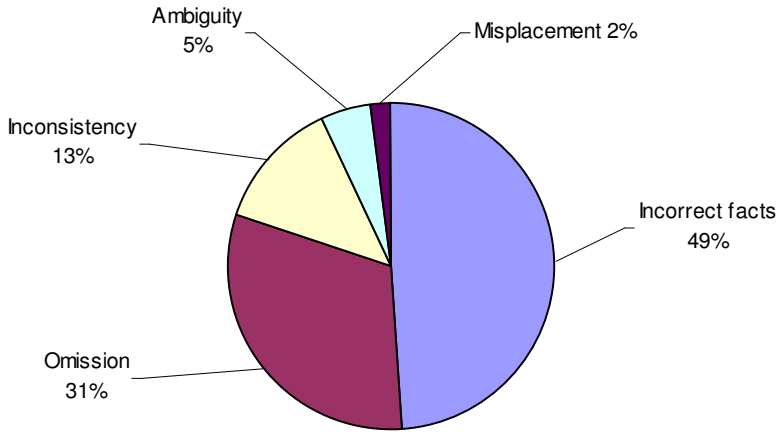



*Figure 1.4 – Distribution of errors in the SRS*

The researchers have been improving the requirements engineering process through various solutions. Developments have been made in the individual phases of the requirements generation process, such as more effective elicitation techniques [Goguen 1993], risk analysis [Davis 1999], verification and validation techniques [McGraw 1997], and so on. In this section we give a brief overview on current approaches that address the above-mentioned problems. The detailed description of the models and techniques discussed here are explained in greater detail in Chapter 2.

Well-known methodologies to coordinate user participation in the requirements elicitation process are Participatory Design (PD) and Joint Application Design (JAD). Both approaches concentrate on facilitated interaction between users and developers, but they differ in participant selection and facilitator roles. PD and JAD are valuable

approaches for requirements elicitation because they focus on group discussions and emphasizing customer input. However, they lack the holistic approach that encompasses the entire requirements engineering process, as well as guidelines for the recording of requirements and ultimately generating the SRS.

The importance of understanding the application domain and analyzing the root cause of the problem is noted by both [Davis 1993] and [Leffingwell 2000]. The problem analysis process includes stating the problem, understanding root causes, and defining the system boundary. Although the detailed steps of the problem analysis process and survey of techniques are provided, the approaches lack the integration of problem analysis to other parts of the requirements generation process. Moreover, the models do not provide guidelines on documents produced during the problem analysis and their content and format.

Documenting requirements is vital throughout the requirements engineering process to ensure correct and complete representation of requirements, and an adequate basis for the future developments of the system. The requirements engineer has a challenging task to incorporate multiple viewpoints from different users and document communicated ideas in an unambiguous and comprehensive way. Two types of requirements representation are (1) natural language and (2) formal specifications. Expressing information in a natural language is easy to comprehend for all parties involved in the development project, but the information expressed can be ambiguous, resulting in multiple interpretations of a particular document. On the other hand, a formal specification overcomes the ambiguities and correctness verification. However, formal techniques are complicated to apply for developers and difficult to understand for the customers [Sutcliffe 2002]. Given the two approaches of representing requirements, natural language is most commonly used. The problems inherent in the natural language description can be satisfactorily overcome with proper guidelines for content and format, and templates for the documents. These aspects are addressed in this thesis.

Most of the current research is focused on individual areas of the requirements engineering process. There is, therefore, the need for a framework that organizes and integrates all requirements generation activities, including elicitation, analysis,

specification, verification and validation, and management. Several models have been recently developed, including the RGM [Arthur 1999], the Knowledge-Level Process Model [Herlea 1999], the Requirements Triage [Davis 1999], and the Win-Win Spiral Model [Beohm 1998]. Each of these models provides structure to the requirements generation process based on a particular approach emphasizing certain areas such as systematic elicitation, importance of scenarios, detailed risk analysis, and requirements negotiation. A significant point often overlooked by these models is the importance of well-defined requirements documents and their evolution throughout the requirements generation process.

## 1.2. Problem Statement

Even though a number of approaches have been developed for requirements generation, software products continue to fall short of meeting the customer needs adequately – poor requirements are most often the cause of this shortcoming [Kasser 1998]. As we analyze the requirements engineering approaches and methods, we observe certain issues that need to be addressed.

- Errors in the SRS – Although different methods and approaches are proposed for generating the SRS, errors still continue to exist. Incorporating changes into the SRS is expensive as changes occur frequently in the software development. To minimize frequent changes, not only is the activity of producing and modifying the SRS important, but the entire requirements generation process from eliciting user needs to generating the SRS needs to be refined.

- Lack of holistic approach – Activities in the requirements engineering have been analyzed and developed on an individual basis. This has lead to a lack of synchronization among the research efforts and a lack of integration among those activities. To address the problem of *adhoc* composition of activities, several models have been recently developed, e.g., [Arthur 1999], [Herlea 1999], [Davis 1999], and [Beohm 1998]. However, these models still need to be refined to provide clear guidelines in the use of methods and the generation of well-defined documents.

This section is focused on the importance of requirements documentation *throughout* the requirements generation process. Requirements documents play a vital role in the software development life cycle as they are the means of communication among stakeholders. They are the principal instrument for understanding, verifying, and validating requirements [Davis 1993A]. However, the current requirements engineering models used in the software industry do not sufficiently reflect the importance of documents.

- Identifying intermediate documents – Existing models do not stress the necessary importance of identifying intermediate requirements documents. The focus of a model is on the flow of process activities, but the transformation of documents is often overlooked. For example, by neglecting problem analysis documents, incorrect requirements can creep into the SRS because of an incomplete understanding of user problems and needs.
- Guidelines on characteristics of requirements documents – Existing models often lack guidance as to the most important characteristics that requirements documents should possess. Some models do mention the representations of the requirements; however, the guidelines and templates for the content and format are not explicitly specified.
- Lack of synchronization between documents and activities – The models do not clearly specify how documents support the objectives of the related activities. This is because in the process of deriving requirements, the intermediate and final embodiments of the requirements lack the necessary characteristics that are needed to support better comprehension and communication. Due to the lack of these characteristics, the document content and format often are not in harmony with the activities defined by the requirements engineering process.

This research focuses on addressing the issues underlying the above discussion: (a) identifying requirements documents that support the objectives of activities, and (b) determining how these documents are transformed in terms of content and format as we transition through the requirements engineering process. The goal of our research is to define the proper form, format, and use of documents to support the requirements

generation activities throughout the entire requirements process. This entails deriving requirements from the user needs and ultimately generating the software requirements specification (SRS). In addition, the research attempts to assist in the accurate recording of requirements, by specifying the templates for appropriate content and format to provide the necessary guidance to the requirements engineer. The synchronization of the documents with the related activities results in a seamless transformation of requirement forms that ensure correctness and a minimal loss of information.

## 1.3. Solution approach

We propose a comprehensive requirements engineering model spanning the entire requirements engineering process. The model is composed of two main parts, (1) a framework of requirements engineering activities at appropriate levels of abstractions, including information about the activity objectives, participants, and strategies on the process steps, and (2) guidelines and templates of the content and format of various requirements representations throughout the requirements engineering process.

In order to develop the proposed model, several issues have to be considered. These issues and the solution approaches are described below:

- Identifying a requirements engineering model and activities – We need a model that encompasses all the requirements engineering phases, and one that is not constrained to a specific application. In addition, the model also must be modifiable so that the flow of requirements can be clearly represented in the model. We choose the RGM model because it possesses the necessary qualities required for our research. However, many activities in the RGM model need to be restructured and modified to reflect the correct level of resolution so that we can more accurately represent the intermediate development of the requirements documents.
- Analyzing activities and their objectives – The objectives of the current activities in the requirements generation models are not well enunciated; therefore, another significant issue is to understand the requirements engineering activities and to identify their real objectives. Upon completion of identifying the activities and

determining the objectives, the requirements model should possess the necessary structure and composition to depict the flow of information through the documents.

- Identifying document characteristics – Various documents are created and used in the process of deriving requirements; hence, it is significant to identify the characteristics of these documents. Hence, the next step in the research is to determine the content and format of the documents produced by each activity in the requirements generation model. We analyze both the documents proposed by the researchers and the ones used in the industry to obtain insights into requirements representation.

- Synchronizing documents and activity objectives – Based on the document characteristics and activity objectives, we need to synchronize the requirements documents with the related requirements engineering activities. The activity objectives direct the content and format of the documents produced by that activity. In addition, input documents required by the activity also need to be in consent with the objective of the activity. Requirements documents are synchronized with the activity objectives to ensure that the objectives are satisfied by the information presented.

- Requirements evolution – A final issue is determining how the requirements captured during the requirements engineering process evolve in terms of content and format as requirements documents are transformed through successive activities. The requirements document change in terms of content and format as they pass from one activity to the other. The tracing of the requirements evolution ensures correctness as requirements are captured, analyzed, documented and verified. By analyzing the requirements documents and the flow of related activities, we identify the evolution process and requirements document transformation as it flows through the requirements engineering process.

## 1.4. Blueprint of Thesis

The remaining chapters of this thesis are organized as follows.

In Chapter 2, we present the background and related work for our research. We emphasize the need for the integrated requirements documentation approach by focusing on the existing SDLC and requirements engineering models, and highlighting the limitations of the requirements documents within those models. Moreover, this chapter also includes a survey of the related requirements engineering approaches and tools, which contribute to this research.

Chapter 3 presents the proposed refinement to the RGM model. Different levels of abstraction are identified, and all activities in the model spanning the entire requirements generation process are explained in details, with an emphasis on identifying and conveying the main objective of the each activity.

Chapter 4 focuses on the detailed description of the artifacts in the requirements generation process. The content and format of each document is determined based on the assessment of different documents used in the industry. In addition, this chapter describes the transformation process of requirements documents, starting from the initial representation of user ideas and going through the generation of the requirements specification.

Chapter 5 presents the summary of this thesis and the contributions made to the requirements engineering field. The possible extensions to this research are also presented as future work.

The appendices provide supplementary information about the research. This section includes templates of the content and format of requirements documents.

# CHAPTER 2. BACKGROUND

## 2. Background

In Chapter 1, we present the rationale for developing a requirements generation model supporting comprehensive guidelines for recording and representing requirements documents. In this chapter, we present the background information that had had a significant influence on this research.

The information presented in this chapter is organized into four parts. In Section 2.1, we present definitions and terminologies as a basis for establishing a better understanding of requirements. The next section (2.2) reviews popular software development paradigms and examines how requirements generation and documentation is used in associated models. The evaluation of software models is followed by Section 2.3, where we describe the major activities of the requirements engineering process and evaluate the existing requirements generation models. Our objective is to identify the best features of the available models by studying their strengths and weaknesses. The last section (2.4) includes a literature survey of the requirements documents produced in the requirements generation process. In addition, several tools supporting the requirements document generation and refinement are discussed in this section.

## 2.1. Requirements

This research addressed the use of requirements to facilitate better comprehension and seamless transformation of information. Thus, requirements are at the basis of our research, and in order to develop a common understanding of requirements, this section presents the definitions and identifies the various types of requirements.

### *2.1.1. Requirements Definitions*

The literature provides a number of definitions of the term “requirement”. IEEE defines requirements as [Macaulay 1996]:

1. A condition or capacity needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

This definition highlights two important characteristics:

1. Requirements are reflections of the user needs.
2. Requirements are abstract descriptions of what the system will do without referring to how it will accomplish it.

The literature includes several other definitions [Thayer 1997], [Sommerville 1996], [Lawrence 1996], and so on.

A distinctive definition of "requirement" has been proposed by [Jackson 1995]:

> *A requirement is a desired relationship among the phenomena of the problem context, to be satisfied by the software/machine.*

According to Jackson, software development is synonymous with building a machine by describing it. He defines requirements as a phenomenon of the application domain, *not* the solution/machine domain [Jackson 1995]. Hence, the requirements specification describes the *external* behavior of the machine, including requirements of the shared phenomena of the application domain and the machine. Therefore, it is necessary to understand both the application domain (problem domain) and machine domain (solution domain) to generate a well-defined requirements specification. The relationships between the requirements, specifications, and both machine and application domain phenomena are illustrated in Figure 2.1.

In the process of deriving the requirements specification, a number of *requirements documents,* which are various representations of requirements, are generated and analyzed. *Requirements evolution* is a process of transforming the content and format of requirements documents as we progress through the requirements engineering process.

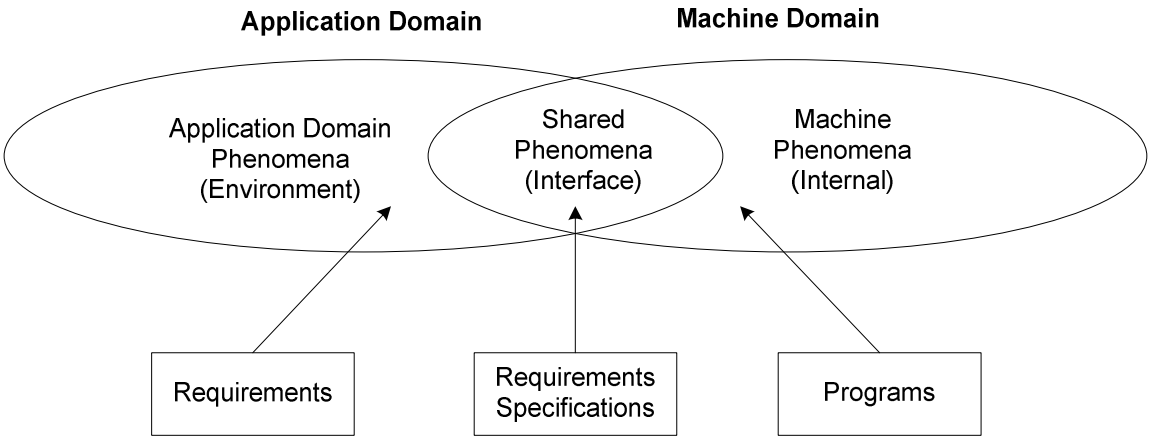



*Figure 2.1 – Relationships of requirements, specifications, and domains*

### *2.1.2. Classifications of Requirements*

Several classifications of requirements have been proposed in the requirements engineering literature. Here, we present a brief overview of how requirements are classified. A detailed analysis of requirements classifications is discussed in Chapter 4.

IEEE classifies requirements into five categories – functional, interface, performance, quality, and design constraints [IEEE 1990]. In addition, different categorizations are proposed by [Davis 1990], [Leffingwell 2000], and [Lauesen 2002]. All these classifications of requirements can be condensed into a single categorization comprising of two types of requirements:

- **Functional requirements** – also referred to as behavioral requirements, functional requirements define precisely what inputs the system expects, what outputs will be generated by the software, and the details of the transformational function existing between those inputs and outputs [Davis 1990]. Some authors classify the input and output requirements as separate from the functions and refer to them as *data requirements* [Lauesen 2002].
- **Non-functional requirements** – define the overall qualities or characteristics of the resulting system. They place restrictions on the product being developed, the development process, and they also specify external constraints that the product must meet [Kotonya 1998]. Non-functional requirements can be further divided

into three sub categories – product, process, and external requirements (Figure 2.2). While product requirements detail the desired characteristics a system or subsystem must possess, process requirements specify constraints on the development process. External requirements are constraints derived from the environment in which the system is developed [Kotonya 1998].

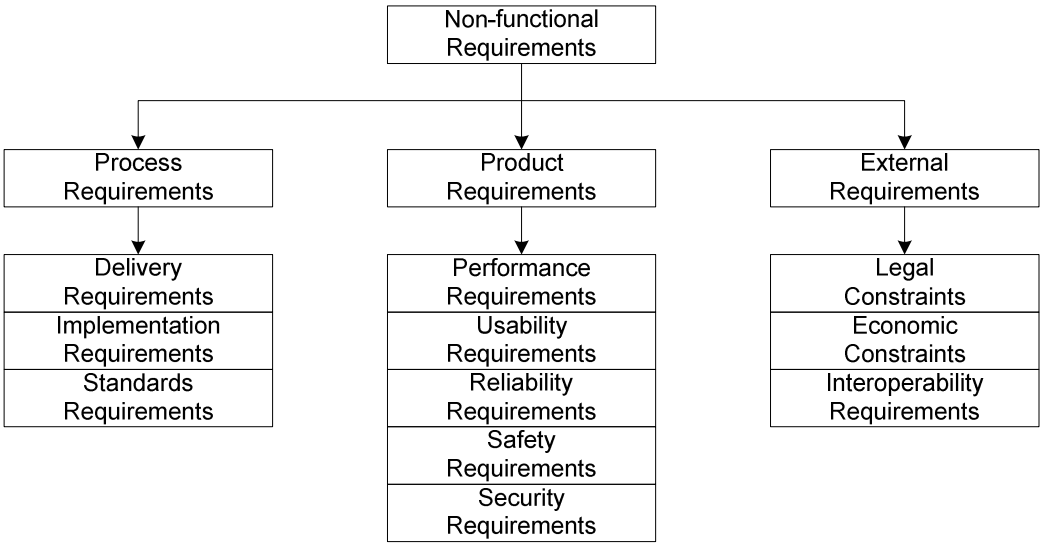



*Figure 2.2 – Categories of non-functional requirements*

Another way of classifying requirements is according to the level of satisfying the customers [Zultner 1992].

- **Normal requirements** – The objectives and goals of a system that are elicited during meetings with the customer/user. If these requirements are implemented, the customer is satisfied.
- **Expected requirements** – These are often so fundamental and basic that the customers may not mention them. Their absence causes a significant dissatisfaction.
- **Exciting requirements** – These features are beyond the customer expectations, and their presence is very pleasing.

All types of requirements presented in this section need to be generated through a structured process to ensure completeness and correctness. The subsequent sections

discuss how requirements generation and documentation is addressed in the existing SDLC models and the requirements generation models.

## 2.2. Requirements in the SDLC models

SDLC spans the complete product life cycle, beginning with the initial conception of the idea, and continuing through to the day the product is withdrawn from the market or service. Ever since Royce introduced the first SDLC model in 1970 [Royce 1970], a number of models have been proposed to improve the software engineering process. This section presents a brief description of the industry accepted SDLC models and concentrates on how these models address the issue of requirements generation and documentation.

### *2.2.1. Waterfall*

The waterfall model [Royce 1970] is known as the basic or "traditional" software engineering model. The model proposes a linear, sequential approach to software development consisting of five phases – analysis, design, coding, testing, and maintenance as shown in Figure 2.3.

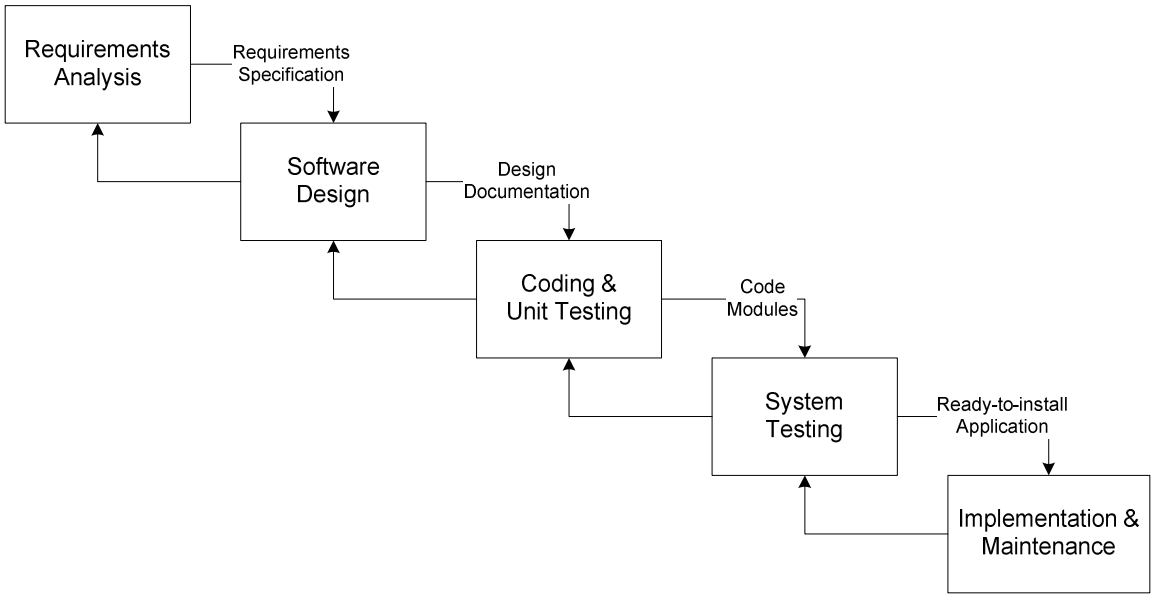



*Figure 2.3 – Waterfall model*

Requirements analysis is the first phase in the waterfall model, and its main objective is determining the scope of the problem and identifying the requirements of the system. The analysis phase begins with a careful study of the project feasibility [Jalote 1999]. Project feasibility is followed by a thorough analysis of the problem and solution space to determine the requirements. The requirements engineer must have a good understanding of the application domain in order to document the required functions, interface, and constraints in the SRS, the final work product of this phase. It is necessary to verify and validate the SRS to ensure that the specification is complete and correct.

The waterfall model provides the much needed structure and framework for successful software development. In addition, this model emphasizes the importance of verification by stipulating the verification activity at the end of each phase. On the downside, this model is criticized because of its non-iterative nature which fails to reflect the real world software development scenario [Hanna 1995]. However, in spite of the criticisms of the waterfall model for being inflexible, variations of this model are still the most popular in the industry [Neill 2003] [Holt 1997].

In the waterfall model, the only requirements document generated is the SRS. Even though the model emphasizes understanding the application domain and eliciting required functions from the users, it does not provide guidelines on how to document them. In addition, this model performs V&V only on the completed SRS, resulting in longer period of checking and correcting the specification.

### *2.2.2. Prototyping*

Prototyping evolved out of the need to overcome weaknesses of the waterfall model and to obtain user feedback early in the development process by conducting user evaluations of a “quick” implementation of the system.

As depicted in Figure 2.4, prototyping begins by gathering initial requirements. Meetings between the developer and customer are conducted to determine overall system objectives, functions, and performance. The preliminary version of the requirements specification document is produced based on the elicitation [Gomma 1981]. The developer then uses a set of tools to implement a quick design and a working model

(prototype) in order to understand the proposed system, or certain aspects of it, and to clarify requirements [Maude 1991]. Prototypes have little or no underlying functionality, with the sole purpose being to capture user feedback in a short time frame. Minimal cost and time is spent to design and refine the prototype since it is not used as a part of the final system [Brooks 1995]. Therefore, the process is often referred to as *rapid prototyping*. The users evaluate the functions and recommends changes to reflect their actual needs. The requirements phase is iterative, and emphasizes redesigning the prototype until an acceptable model is derived and the SRS is generated. Based on the prototype and the SRS, the developer builds the real product by applying the steps described in the waterfall model.

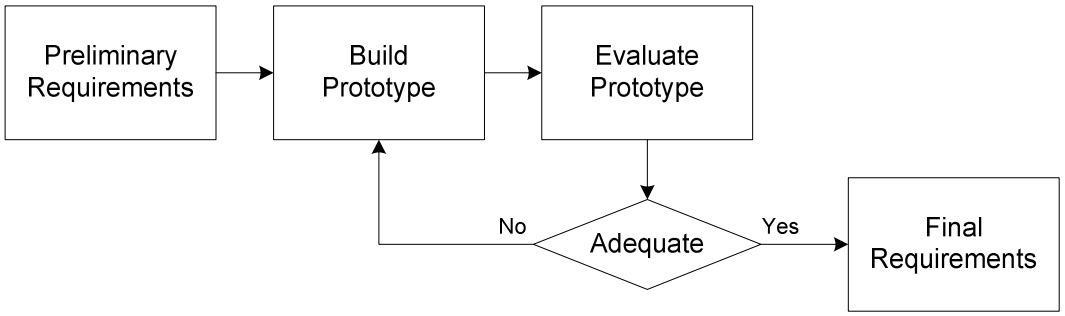



*Figure 2.4 – Rapid prototyping model*

An alternative way of using prototypes is *evolutionary prototyping*, where the prototype is iteratively refined and becomes the final product. Thus, the prototypes are not discarded as in the rapid prototyping process. Through multiple iterations the prototype is expanded and polished to obtain the product satisfying the user's needs. A difficulty of evolutionary prototypes is keeping the code efficient and error-free, while continuously adding new code or making changes to the existing code.

The prototyping model emphasizes understanding the user requirements, especially user interface and transaction-oriented functions, to generate a quality SRS [Dorfman 1997]. Prototyping allows the software engineer to refine requirements that are not well understood. Therefore, the prototyping approach addresses the volatile nature of the requirements, and through iterations, reduces risk of developing the wrong product

[Jalote 1999]. Furthermore, prototyping enables the creation of a specification with precise details about the user interface and functionality of the system. In addition, prototyping is also very useful for eliciting requirements for systems which have not been identified earlier.

However, demonstrating an early design without analyzing requirements in detail can cause unnecessary design constraints. Another inherent problem is when the user sees the prototype, what appears to be a fully working system, he/she mistakenly believes that the prototype can be easily transformed into a final product. Therefore, users tend to demand a short duration for releasing the working product. Moreover, a risk of producing "spaghetti code"[1] is apparent in the development is evolutionary prototypes. Developers often make implementation compromises in order to get a working prototype rapidly. This may result in inappropriate design decisions and inefficient algorithms. Thus, effective management of the prototyping model is crucial.

### *2.2.3. Incremental*

The incremental development combines features of the sequential model with the iterative philosophy of prototyping. The software product is designed, implemented, integrated, and tested as a series of incremental deliverables which build on the previous increments of the software [McDermid 1993]. The first increment is usually the core product, which implements only the basic requirements [Pressman 2001]. The subsequent increments implement the additional features of the system. After the development of each increment, the product is evaluated, and a plan for the next increment is created. The plan addresses the modification of the core product and the delivery of additional features and functionality. This process is iterated following the implementation of each increment, until the complete product is developed. The iterative behavior of the incremental model is illustrated in Figure 2.5.

[1] Code which is inefficient, confusing, and error prone.

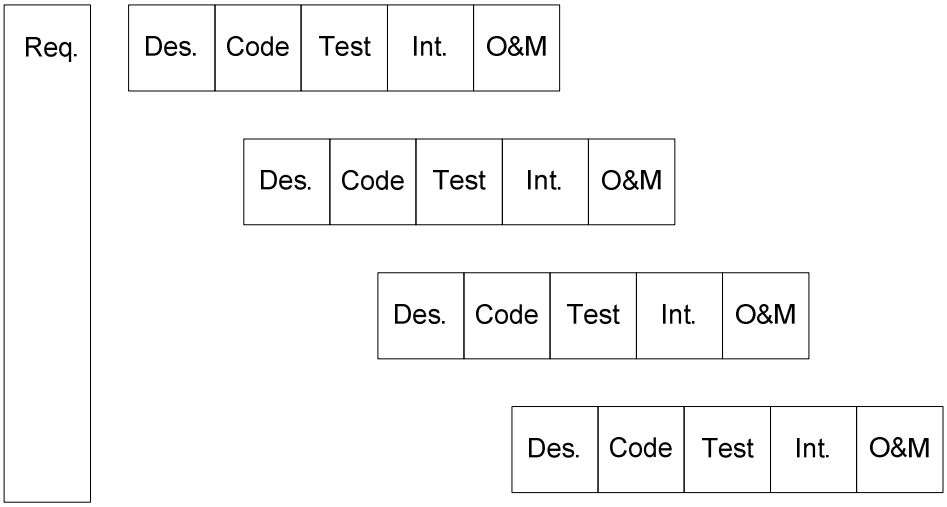



*Figure 2.5 – Incremental model*

The requirements engineering phase in the incremental model produces a single requirements document – the SRS. In the original concept of the incremental model, requirements are assumed to be stable [Dorfman 1997]. However, this concept is altered by allowing the SRS to be modified in each increment, since in practice requirements in the later phases can be changed based on evaluation of earlier increments or technology advancement [Dorfman 1997]. Therefore, the SRS is produced in increments, with the initial specification containing detailed requirements to be implemented in the first increment. Features to be included in the subsequent increments are noted in the SRS but not detailed. After each increment, the feedback obtained from the users is collected and analyzed for revision to the SRS. In addition, new features can be included as a result of the user needs.

The incremental model emphasizes continuous user feedback to obtain the SRS. Even though this model is iterative like the prototyping model, the incremental model focuses on the delivery of an operational product with each increment. This model is specially suited for situations where the required staff is not available for a complete implementation by the deadline established from business needs [Pressman 2001]. The incremental model lowers the risk of project failure by allowing partial delivery of the system. Determining the number of iterations and its duration is very important to the

success of the project. If the software product is broken down into too few builds, the model becomes a build-and-fix approach. On the other hand, if the project has too many builds, it can incur excessive overhead. In addition, the requirements change management process is not explicitly represented in this model.

### *2.2.4. Spiral*

The spiral model as defined by [Boehm 1988] consists of six task regions: customer communication, planning, risk analysis, engineering, construction and release, and customer evaluation. Each phase in the development life cycle iterates one or more times through the six sections. The objectives of each region are given below [Pressman 2001]:

- Customer communication – determines the scope and requirements with active participation of the customer.
- Planning – determines objectives and possible solution approaches.
- Risk analysis – along with the customer, determines the risk involved in each alternate solution approach.
- Engineering – creates the design of the system and describes the necessary algorithms.
- Construction and release – codes the application based on the design, and conducts testing and release.
- Customer evaluation – determines if product is satisfactory and what changes or changes are necessary.

The spiral model (Figure 2.6) combines the best features of the waterfall and prototype models, and includes the evaluation of risk. Before each development phase begins, the risks involved in the project are analyzed and mitigated, if possible. If the risks cannot be resolved, then the project may be discontinued by the decision of the management [Schach 1996].

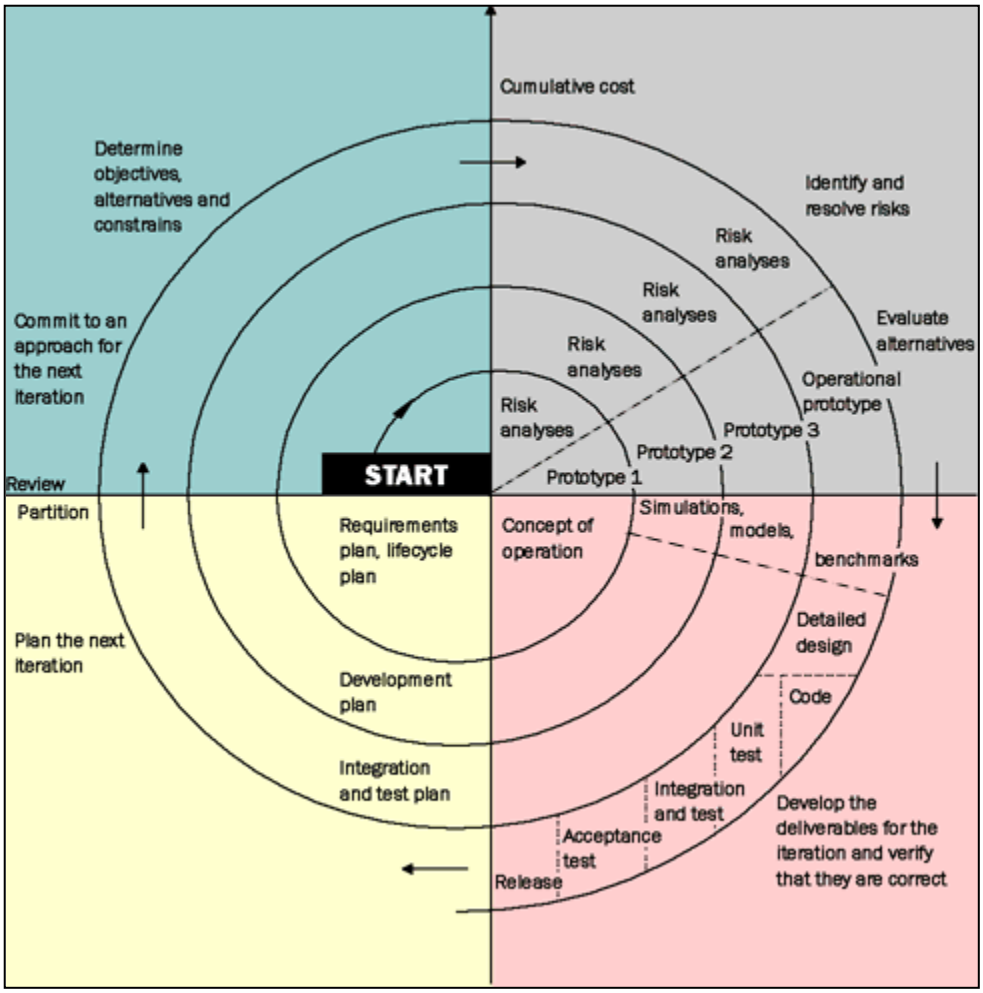



*Figure 2.6 – Spiral model*

The spiral model uses prototyping as a means of reducing the risk in the project. Each of the cycles in the spiral model is iterative and can be executed several times until the objectives are achieved. The SRS is the final product of the requirements phase, and it is reviewed and validated several times until it is complete. In the initial cycle, only the known requirements are elicited and prototyped. The risks are evaluated and the plan for the next iteration is prepared. Before conducting the next cycle, the requirements are validated by the customer, and the feedback is used to make any needed modifications to the SRS. Thus after several iterations, a complete and precise SRS is generated.

The spiral model focuses on eliminating errors and unattractive alternatives early in the development process through the use of risk analysis. Defining the system through an

iterative approach allows the stakeholders to remain focused on the smaller manageable issues within each cycle. Thus, this model is better suited for large projects as compared to the other models. A strong focus of the spiral model is on communications with the customer. The software engineer works together with the customer to (1) decide on solution alternatives, (2) determine the risk of the system/solution, and (3) evaluate the delivered system.

The disadvantage of the spiral model is that the process is not well documented and fails to provide clear guidance to the requirements engineer. The model needs further elaboration of each spiral, and lacks detailed checklists and guidelines necessary for consistent interpretation and use of the spiral model. Furthermore, risk analysis is a vital part of the process; therefore, the success of the project is largely dependent on the risk-assessment expertise.

### *2.2.5. XP*

The main objective of XP (Extreme Programming) is to shorten the development time of a project while increasing the interaction with and feedback from the customer [Beck 1999]. XP proposes a core set of principles – small releases, pair-programming, continuous integration, and on-site customer participation.

Communication and customer involvement is a major component of XP. In fact, XP takes customer interaction to the extreme by including customer participation in all phases of the development life cycle. XP also stresses on product evaluation, which is accomplished by writing and performing test cases (Figure 2.7). Another important principle of XP is pair programming. In pair programming, two developers always work together, one developer on the keyboard and the second giving instructions or simply reading over the shoulder of the typist to ensure coding accuracy.

The requirements engineering phase in XP occurs as the requirements are elicited from the on-site customer, who takes part in the complete development life cycle [Beck 1999]. Once the requirements are elicited, the developers then ask the customer to prioritize the requirements, which are implemented according to their priority. The test cases for the prioritized requirements are then prepared by the customer with help from the developer.

Finally, the product is coded and validated against the test cases by the customer. The feedback obtained from the customer is used in the planning of the next increment. No formal requirements document such as SRS is produced in XP since documentation is considered to cause delays to the project schedule.

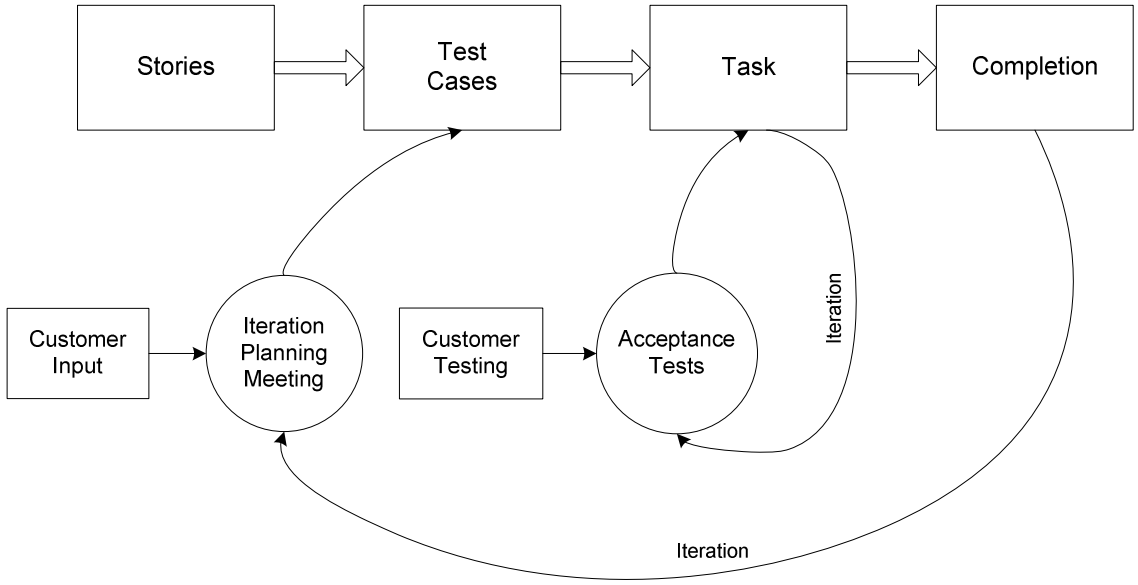



*Figure 2.7 – Extreme Programming Model*

XP is guideline centric and hence lacks detailed descriptions for producing stories and tasks [Leite 2001]. The process follows the evolutionary approach of building the product in increments, and thus improving the confidence of the customer in the project. Since the customer is involved at all times in the XP development cycle, the finished project has a better chance of meeting the customer expectations. However, assuming that all business domain concerns can be represented by only one on-site customer is impractical, especially for complex systems [Leite 1998]. XP offers faster completion times for the project by avoiding documentation overhead. Even though the development time is shortened by less documentation, problems are faced in maintenance due to lack of reference documents.

**Summary:** On analyzing the various SDLC models discussed in the previous sections, it is clear that all the models are focused on the end product of the requirements engineering process. Even though the general approach is described in the SDLC models,

no detailed methods or guidelines for generating requirements are provided. Moreover, how the SRS is derived from the elicited user needs is not explained. The SDLC models provide a general description of the process for generating the SRS, but fail to explain the intricacies of the requirements engineering process.

Although SDLC models fail to provide the details of the requirements engineering process, they do place the requirements generation process in perspective and relative to the rest of the software development process. For example, the usage of requirements documents in other software development phases and changes to the requirements due to developments in other phases are addressed. An abstraction of the SDLC gives a high-level overview of the project, which allows for the partitioning of the project into smaller, well-defined phases (a structure) that are easily manageable. We note that SDLC models leave the activities within the phases open to the software engineer's discretion; it doesn't provide detailed guidelines on how to produce requirements.

## 2.3. Requirements Engineering Process

In the previous section, we highlighted the fact that the requirements phase in the SDLC lacked the necessary structure to produce a complete and correct SRS. This section presents research efforts intended to improve the requirements engineering process. First, we describe the basic activities that need to be performed for a successful requirements engineering process, and then present various requirements engineering models and their approach to the generation of requirements.

### *2.3.1. Requirements Generation Activities*

Formerly known as "requirements analysis" in the SDLC, requirements engineering is composed of several activities, which have been named differently in the literature. Jarke and Pohl identify three phases in the requirements engineering process – elicitation, expression, and validation [Jarke 1994]. Davis describes two distinct requirements engineering phases, referred to as problem analysis and product description [Davis 1990]. Similarly, Sawyer proposed a four phased generic model, consisting of discovery, analysis, negotiation, and specification [Sawyer 1997]. The literature also includes many

other classifications of the requirement engineering activities, such as in [Krasner 1989] and [Leite 1991].

Thus, we see that the requirements engineering research does not define the requirements engineering phases uniformly. However, in this thesis, the requirements engineering phase is considered to be an iterative process consisting of five main phases – elicitation, analysis, specification, verification, and management [Thayer 1997]. The requirements generation process starts by eliciting user goals and needs. Elicited information is then analyzed and modeled to describe the requirements, which is then formally represented by the requirements specification (SRS). The specification is then evaluated for correctness and completeness. Management is a continuous process to maintain the integrity and consistency throughout all requirements generation phases. In following sections, each phase is described in detail.

#### 2.3.1.1. Requirements Elicitation

Requirements elicitation is an iterative process of discovering and articulating the user needs for a system by communicating with the stakeholders, including customers, system users, developers, and other affected parties [Sawyer 1997].

In order to define the functions and attributes of the new system, a comprehensive knowledge of the environment (application domain) is required. The information about the application domain can be elicited from many sources such as stakeholders, organizational documentation, and the existing software system. A stakeholder is anyone who has direct or indirect influence on the system requirements [Kotonya 1998]. Identifying all stakeholders is an important part of elicitation, as different viewpoints of the stakeholders need to be captured for further analysis and negotiation.

Compared to other phases in the requirements engineering process, elicitation is the most communication intensive phase; therefore, the elicitation techniques are related to the social sciences, organizational theory, and group interaction research [Potts 1991].

A variety of techniques are utilized for requirements elicitation, such as interviews [Gause 1989], scenarios and use cases [Weidenhaupt 1998] [Jacobson 1992], goal-based approaches [Potts 1994], and brainstorming. Goguen proposes different elicitation

techniques such as introspection, observation of the existing system, protocol analysis, and ethnography [Goguen 1993]. The viewpoint approach [Sommerville 1998] introduced in recent years is another promising technique attempting to capture and negotiate different views of stakeholders. A number of elicitation techniques are also described by [Lauesen 2002] and [Christel 1992].

While requirements elicitation is considered the earliest activity in the requirements engineering process, it can not be separated from the subsequent activities. Elicitation is performed iteratively with other activities such as local analysis and prioritization during the requirements generation process [Christel 1992].

#### 2.3.1.2. Requirements Analysis

The requirements analysis is primarily concerned with understanding the nature and scope of the requirements to assess the possible risks involved in satisfying them [Hull 2002]. In other words, Requirements Analysis is the process of analyzing user needs to arrive at a definition of software requirements and ensuring that a correct software product is being built [Brackett 1990].

Once requirements are elicited, requirements analysis phase is conducted with following objectives:

- Modeling of the requirements
- Creating requirement groups based on functionalities
- Evaluation of the requirements for project factors such as risk and feasibility

Modeling is an important objective of this phase, and it involves the basic issues of representing and reasoning about the knowledge and information captured during the elicitation phase [Greenspan 1994]. Greenspan *et al*. were among the first who recognized the importance of modeling the requirements and argued that analysis is needed to understand and represent the requirements in a clear and comprehendible manner [Greenspan 1982]. Yeh *et al*. state that the complexity of large computer-based systems demands an additional layer of understanding between the real world and the requirements specification [Yeh 1984]. This is accomplished through modeling, which is

generally performed by starting with the representation of the proposed system's environment, followed by working towards the system and its software component.

The models are also used for evaluating the system requirements on the basis of various requirement attributes [Nuseibeh 2000]. During analysis of the requirements, the attributes, such as status, priority, effort, and risk, are determined for each requirement. These attributes help in the evaluation of the requirements during the analysis activities – risk analysis, cost and schedule analysis, market analysis, and feasibility analysis. It is important to understand the risks associated with implementing each requirement, including conflicts with other requirements, dependencies on external factors and technical obstacles. The analysis of requirements from various perspectives enables one to determine if the project is feasible for development.

Analysis also involves interacting with the users to clarify misunderstandings and determine the priorities and value of requirements. In practice, requirements are negotiated, not just elicited; it is incorrect to assume that complete and well-defined requirements for a system are waiting to be discovered [McDermid 1994]. The analysis phase facilitates a common understanding among all stakeholders about the proposed system [Wiegers 2001].

#### 2.3.1.3. Requirements Specification

Many researchers and practitioners claim that the main problems facing project teams are communication barriers and agreement about the contents of the SRS document [Potts 1994]. The argument is that concepts clear to one community of participants can be entirely different to members of another group of participants. The specification phase tries to overcome this barrier of misunderstanding by organizing and recording all the collected information in a precise and understandable document referred to as a software requirements specification (SRS).

The SRS plays a crucial role in the development of software as it is the basic instrument for communication among clients, end-users, system designers, and developers of the software. A good SRS should provide a beneficial platform for:

- Establishing the contract (lists features of the proposed product) between the customers and the suppliers. This contract is binding to all parties and can be used in court as an evidence of what was agreed to.
- Providing a basis for estimating cost and schedule, project planning and management activities.
- Providing a baseline for verification[2] and validation[3].
- Defining the specification for the design phase.
- Reducing the development effort – Careful capturing and documentation of requirements reduces redesign and recoding in the subsequent development phases.
- Providing a basis for documenting user manuals.
- Tracing the requirements to the needs, and vice versa, to support change management of the requirements.

Thus, the SRS represents the user requirements, serves as a contract between the development team and the customer, and provides information to the developers about designing and implementing the system. This document must be internally consistent, complete with respect to the user needs, clear to the stakeholders and developers, and capable of serving as a basis for design and testing phases [Davis 1993A]. The most prevalent standard in the industry is defined by IEEE; it describes the structure, quality, formality, and contents of the SRS [IEEE 1993].

It has been observed that the inadequacies of typical SRS documents – missing, ambiguously presented, or misinterpreted information, poor representation, and obsolete information – are the critical problem characteristics facing the requirements engineer [Levene 1982]. Representation and notation of requirements is a major issue that needs to be addressed for clarity and non-ambiguity. The literature includes a variety of formats and styles, including text and graphics, for the presentation of the SRS. The inclusion of

[2] Process of determining whether the intermediate work products and the final product satisfies the requirements
[3] Process of determining whether requirements satisfy user intent

graphical representations facilitates easier comprehension of the SRS, and thus faster agreement of the contract by the customer. In addition, the SRS can be in a form which enables automatic checking of the consistency of requirements. This has resulted in the definition of several formal requirement specification languages such as PSL/PSA5, PAISLey [Zave 1991] and SDL [Rockstrom 1982].

Another challenge in requirements specification is concerned with the transformation of informal, incomplete, and possibly inconsistent user needs to a precise, unambiguous and consistent specification that can be understood and agreed to by all stakeholders. Recent research by Bubenko *et al*. provides a starting point to address some of the related problems [Bubenko 1994]. However, the research is still incomplete and fails to provide the complete picture of the process transforming needs to formal requirements specification.

#### 2.3.1.4. Requirements Verification & Validation

During requirements verification and validation (V&V), the specification documents are analyzed for compliance with the customer requirements, and the existence of inconsistencies, ambiguities or omissions. Verification ensures the quality of the SRS while validation ensures that the SRS meets the users' intent.

Requirements V&V is a two-part process. The first part determines whether the specification is ready to proceed into design with a high degree of confidence; while the second part ensures that the customers' intent is accurately captured. V&V activities check the requirements for accuracy, completeness, and other desired quality characteristics [Wiegers 2001]. Verification is usually conducted through a formal inspection of the requirements document, while informal reviews are most effective for validation.

The steps involved in V&V ensure that [Wiegers 2001]:

- The SRS correctly describes the intended system behavior and characteristics
- The software requirements are correctly derived from the system requirements, or other origins

- The requirements are complete and of high quality
- All views of the requirements are consistent, and finally
- The requirements provide an adequate basis to proceed with product design, construction and testing.

Several techniques are available for achieving the objectives of the V&V phase. Technical reviews provide an effective approach to conducting requirements verification and validation. Another technique is informal review approach [Wiegers 2001] which involves the distribution of the product to peers for assessment. Informal reviews are good for educating other people about the product and getting unstructured feedback. Compared to informal reviews, which are performed in an *ad hoc* fashion, formal reviews follow a well-defined process with a prescribed sequence of steps. A formal review results in a written report that identifies the material reviewed, the reviewers, the review team's comments, and a summary of defects and issues found.

Kotonya and Sommerville state that several techniques that can be used for validation [Kotonya 1998]. They include:

- Requirements reviews, where a team of reviewers systematically analyze the requirements.
- Prototyping, where end users get a working prototype to test [Lugi 1993].
- Test-case generation, where tests are constructed to see if the requirements are testable.
- Automated consistency analysis, where case tools can be used to check for consistency if the requirements are expressed in an appropriate way.
- Scenarios [Maiden 1998], simulation [Lerch 1997], and animation [Siddiqi 1997].

Requirements V&V offers the biggest potential saving to software development effort, as compared to V&V conducted in design, coding or testing phases. It can identify and correct many errors that otherwise can go undetected until late in the development cycle, where correction would be much more expensive.

#### 2.3.1.5. Requirements Management

Requirements management involves establishing and maintaining agreement with the customers on the requirements for the software project. The agreement forms the basis for estimating, planning, performing, and tracking the project's activities throughout the software life cycle. In other words, requirements management is the set of procedures that assist the overall management of the entire requirements engineering process. In addition, requirements management also controls the constant changes to requirements in the software development life cycle, and ensures that the requirements are consistent.

The key to requirements management is *communication* as it fosters a sense of involvement and ownership among the stakeholders. For the developers to fully understand the needs of the customers, they must have an open channel of communication with the customers. Communication is also crucial when requirements change, as it is important for the changes to be agreed upon.

Requirements traceability forms the core of requirements management. Gotel defines requirements traceability as [Gotel 1994]:

> *…the ability to describe and follow the life of a requirement in both a forward and backward direction (i.e. from its origin, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases).*

Requirements traceability is vital for change control and impact analysis because it shows the dependencies in requirements and identifies potential impact of proposed changes. Effective change management requires a process for proposing changes and evaluating the potential impact of the change on the project [Wiegers 2001]. Any change to the SRS has to be accepted by the customer, and conveying the impact of these changes requires good communication and persuasion skills. Frequent baselining of the SRS is a common practice to ensure that changes to requirements are made to the right version of the specification. In addition, requirements management is often supported by tools that automate the requirements generation and change management activities. Discussion of some of the management tools used in the industry is provided in Section 2.4.2.

**Summary:** These five activities discussed in this section (2.3.1) are essential components of a well-defined requirements generation process. However, for the requirements generation process, we require not only the identification of the activities but also the sequence and order in which these activities are performed. The next section presents the various requirements generation models proposed in recent years, and focuses on how these models attempt to provide structure to the requirements engineering process.

### *2.3.2. Requirements Generation Models*

The literature presents descriptions of several requirements engineering models that strive to provide a framework for the requirements engineering process. In the subsequent sections we discuss some of the models which have influenced our research.

#### 2.3.2.1. RGM

RGM (Requirements Generation Model) proposes a refinement to the conventional requirements generation process by providing a structured framework to support the accurate capturing of requirements [Arthur 1999]. Besides structuring the activities to elicit, articulate, understand, and review the user's requirements, the RGM includes a monitoring methodology which ensures that the elicitation activities are being followed correctly.

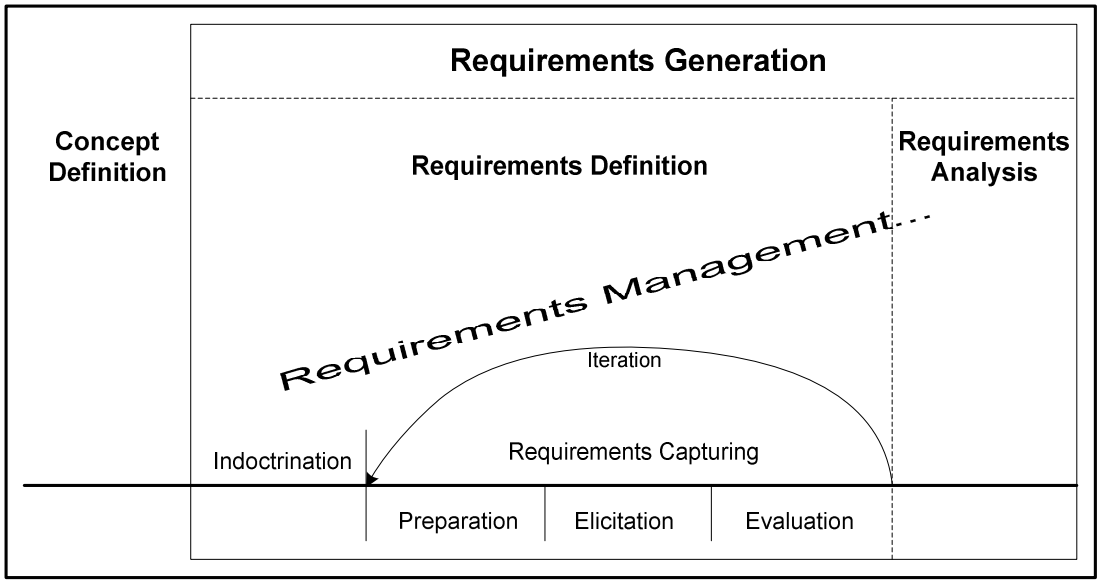



*Figure 2.8 – Requirements Generation Model*

The RGM has refined the requirements capturing phase into preparation, elicitation, and evaluation as shown in Figure 2.8. Prior to conducting these activities, RGM recommends performing the indoctrination activity, whose objectives are to:

- Familiarize the customer to the requirements definition process
- Provide the requirements engineer with an overview of the user's problem domain and needs
- Describe the participants' responsibilities in the requirements engineering process.

The requirements capturing phase along with the indoctrination activity is collectively called requirements definition. The objective of the requirements capturing phase is directly related to the goals of its sub-phases – preparation, elicitation, and validation.

- **Preparation** – defines the scope of the elicitation meeting and ensures that all participants have completed their pre-meeting assignments.
- **Elicitation** – enables the requirements engineer to accurately identify and record software requirements as expressed by the customer.
- **Evaluation** – evaluates the output of the elicitation meetings, identifies unresolved (or new) issues, and determines if additional refinement iteration is needed.

The three sub-phases are iterative in nature, and encourages progressive identification, refinement, and elaboration of individual requirements. In addition, guidelines and protocols are assigned to these activities to focus on capturing requirements. Guidelines are suggestions or recommendations that offer support, whereas protocols are rules that establish boundaries through pre-defined constraints. RGM also supports the management of activities through its monitoring methodology, which detects divergences in the prescribed elicitation process.

RGM has identified the major requirements engineering phases – elicitation, analysis (includes specification), verification, and management. The focus of the model is on decomposing and explaining the requirements generation activities. However, the

characteristics of the intermediate work products generated in the requirements engineering process are not articulated. In addition, the RGM lacks guidelines for the documentation of requirements and fails to address how requirements documents evolve during the requirements generation process.

#### 2.3.2.2. Requirements Triage

The Requirements Triage model [Davis 1999] recognizes the importance of an analysis process to decide which features are appropriate for inclusion in the product. Hence, by refining the analysis phase, the Requirements Triage determines a set of features that can be implemented with acceptable risk levels and marketed at a profitable price with a reasonable return-on- investment.

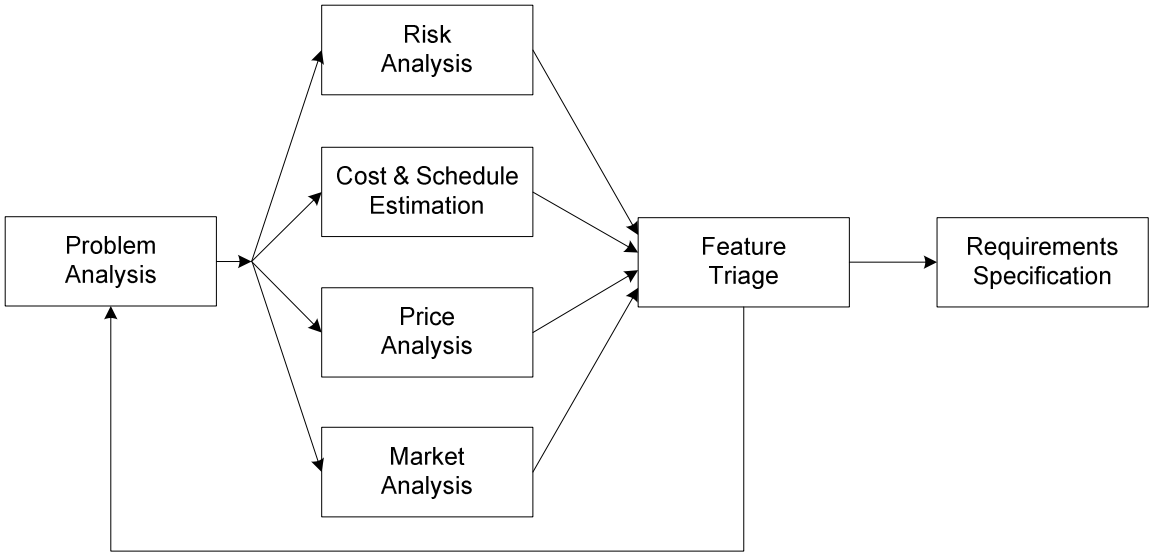



*Figure 2.9 – Requirements Triage Model*

As depicted in Figure 2.9, the Requirements Triage is composed of three phases – problem analysis, requirements triage, and requirements specification. The analysis and specification phases are identified to illustrate how the Requirements Triage model fits into the requirements engineering process. The model provides a brief description of these phases while the emphasis on the requirements triage.

- **Problem analysis**: It involves understanding the needs (*features of the system)* of the stakeholders, and collecting them in a repository for future analysis.
- **Requirements Triage**: The goal of this model is to decide precisely what is to be included in the development of the product so that the organization makes profits. This model considers marketing, development, and financial factors to decide what product is to be built. The Requirements Triage advocates the consideration of factors such as price, market, cost, time-to-market, market size, feature mix, revenues, profits, and return-on-investment as a key to project success. If any of these factors indicates that the feature is a losing proposition, it is advisable that the feature be left out of development. The Requirements Triage also prioritizes the features so that the more important requirements are developed first. Features which are not included in the current development are listed for further releases.
- **Requirements Specification**: It documents the features obtained by the Requirements Triage. The objective of this phase is to ensure that customers, developers, and management have the same understanding of what is to be built.

The emphasis of the Requirements Triage model is on the analysis phase; the other activities are neither refined nor explained in detail. The requirements triage incorporates an elaborate, yet practical requirements analysis methodology. Evaluating the features of the product from multiple standpoints facilitates verification and validation, thus ensuring that the product is feasible. The Requirements Triage recognizes an intermediate document to the SRS – the prioritized list of requirements ranked according to cost, schedule, risk and price.

#### 2.3.2.3. Knowledge-Level Process Model

The Knowledge-Level Process Model advocates a scenario-based requirements engineering approach, in which requirements and scenarios are considered equally important [Herlea 1999]. This model includes several abstraction levels and expresses the requirements in various degrees of formality.

The Knowledge-Level Process Model identifies 3 phases – elicitation, manipulation of requirements and scenarios, maintenance of requirements and scenarios (Figure 2.10).

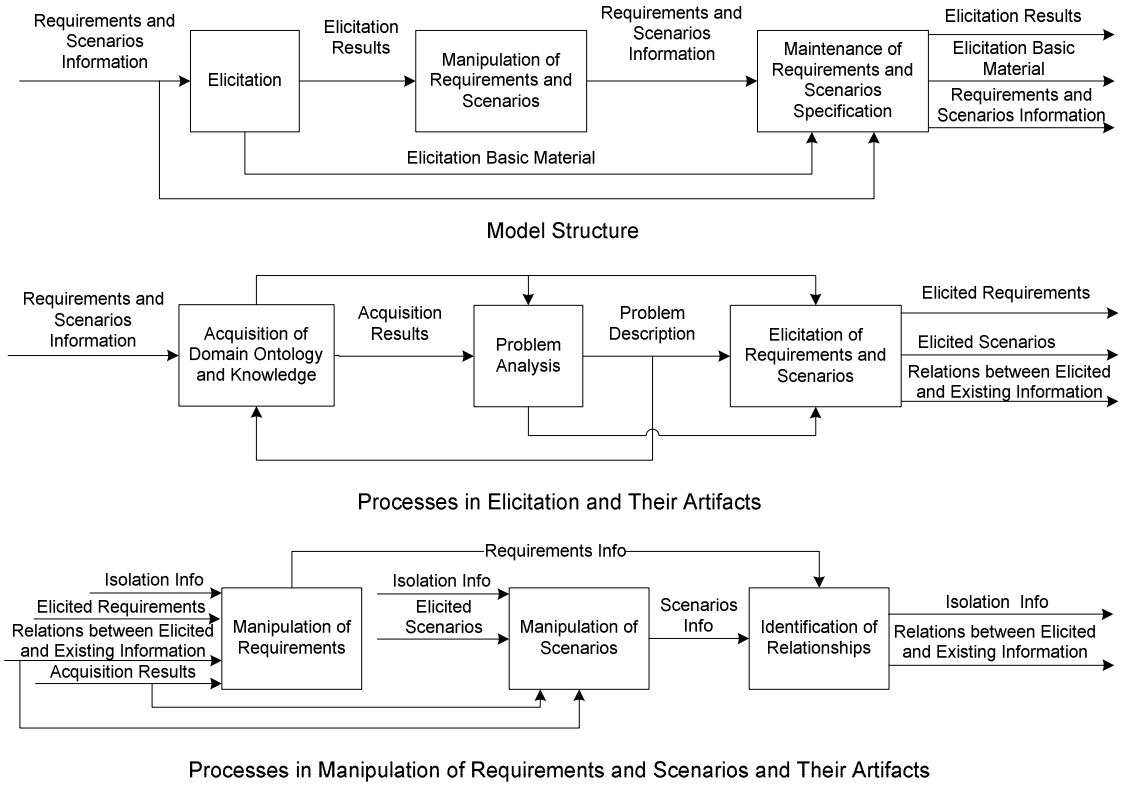



*Figure 2.10 – Knowledge-Level Process Model*

- **Elicitation**: This phase provides initial problem descriptions, requirements, and scenarios elicited from stakeholders, as well as domain knowledge. The Elicitation phase is decomposed into 3 sub-phases [Herlea 1999]:
    - Problem analysis – elicits the perceived problem from the stakeholders.
    - Knowledge acquisition – elicits ontology information and knowledge of the domain from stakeholders.
    - Elicitation of requirements and scenarios – obtains requirements and scenarios from stakeholders based on identified problems, existing requirements, and scenarios.
- **Manipulation of requirements and scenarios:** The objective of this phase is to resolve ambiguous, inconsistent requirements and scenarios. This phase reformulates requirements from informal requirements and scenarios, to a more

structured, semi-formal set of requirements and scenarios, noting the relationships among requirements and scenarios. Towards the end of this phase the requirements and scenarios are validated by the stakeholders and clusters of related requirements are identified based on the clustering criteria.

- **Maintenance of requirements and scenarios:** This phase is used to store and retrieve the documents in which the information pertaining to requirements, scenarios, and traceability are represented.

A positive feature of this model is that it provides a useful approach to process modeling through "separation of concerns." The phases of the model can be treated separately and be decomposed into its constituent sub-processes or activities, facilitating a deeper understanding of the process as a whole. In addition, several layers of abstraction can be customizable according to the customers needs. Clear guidelines for elicitation and transformation of requirements and scenarios are provided, and the flow of input and output documents is also shown. However, even though the model identifies intermediate requirements documents, it does not elaborate on the format or content of these documents. Another disadvantage is that the model does not address the non-functional requirements.

#### 2.3.2.4. SCRAM

SCRAM is another scenario-based requirements generation model which (1) produces informal and formal representations of scenarios, and (2) captures the relationship between scenarios, specifications, and prototypes [Sutcliffe 2002].

As shown in Figure 2.11, the model identifies four phases for the requirements generation process – (1) initial requirements capture and domain familiarization, (2) storyboarding and design visioning, (3) requirements exploration, and (4) prototyping and requirements validation [Sutcliffe 2002].

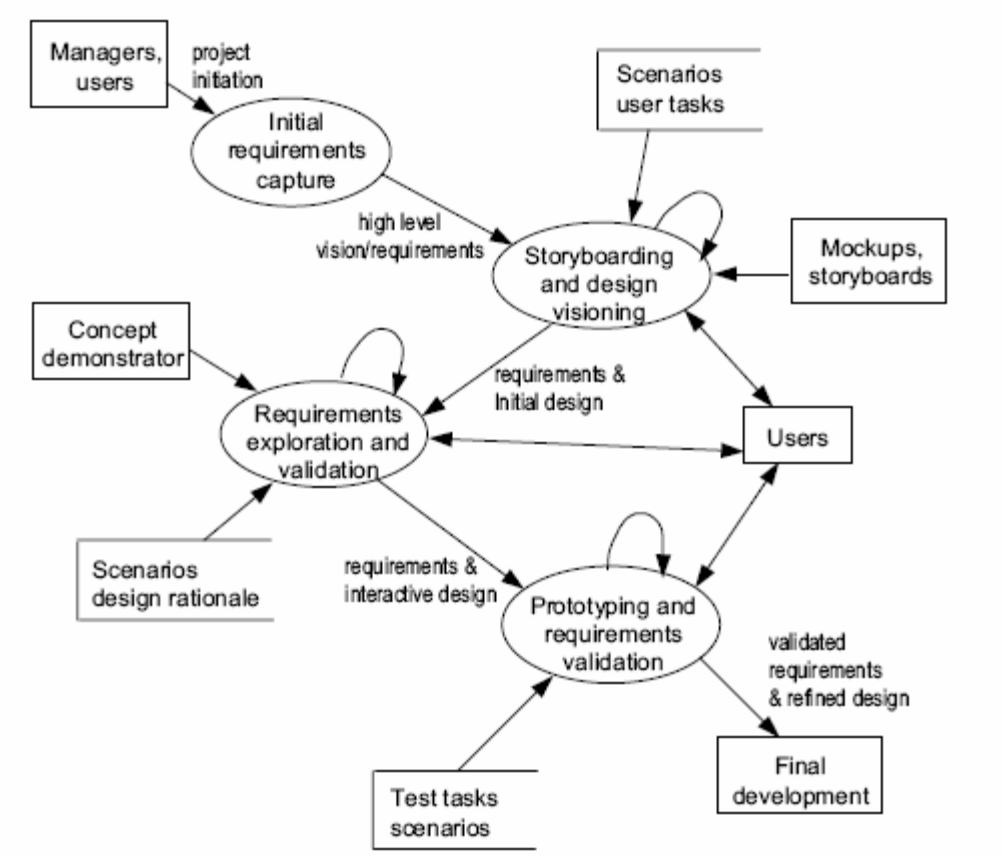



*Figure 2.11 – SCRAM Model*

- Initial requirements capture and domain familiarization – The objective of this phase is to gain sufficient information from the stakeholders to develop the initial definition of the problem and needs. Information about the domain and high level goals of the system are identified in this phase.
- Storyboarding and design visioning – This phase creates early visions of the required system, which are explained to the users through storyboards and walkthroughs to obtain user feedback.
- Requirements exploration – Uses concept definitions and early prototypes to demonstrate the detailed design of the system. Scenario-driven, semi-interactive demonstration is the commonly used technique employed in this phase [Sutcliffe 2002].

- Prototyping and requirements validation – This phase develops more fully functional prototypes and continues refining requirements until a prototype is agreed to be acceptable by all the users.

The SCRAM model emphasizes the constant customer feedback and iterative scenario-based requirements elicitation. The model provides guidelines for elicitation, verification, and validation activities in the general requirements engineering process. However, no explicit description of the specification, analysis, and management phases are discussed. Although SCRAM identifies informal and formal representations of scenarios, the representations of the requirements are not explained in depth. In addition, activities identified in the SCRAM need to be decomposed further to establish a uniform and clear understanding of the activities.

#### 2.3.2.5. Win-Win Spiral Model

The Win-Win Spiral model was developed to overcome the shortcomings of the earlier spiral model by recognizing that conflicts arise among stakeholders and developers during requirements elicitation process. Hence, the model emphasizes negotiations between the stakeholders and the developers [Beohm 1998], and augments each cycle of the spiral model with a set of negotiation activities as follows:

- Identification of the system or subsystem's key "stakeholders".
- Determination of the stakeholders' win-conditions, which reflect the customer's needs.
- Negotiation of the stakeholders' win-conditions.

The Win-Win requirements generation cycles also include activities assessing the feasibility of the project, defining operational concept[4], identifying the risks, evaluating alternatives, developing prototypes, generating software requirements, and finally reviewing the requirements (Figure 2.12).

[4] top-level system objectives and scope

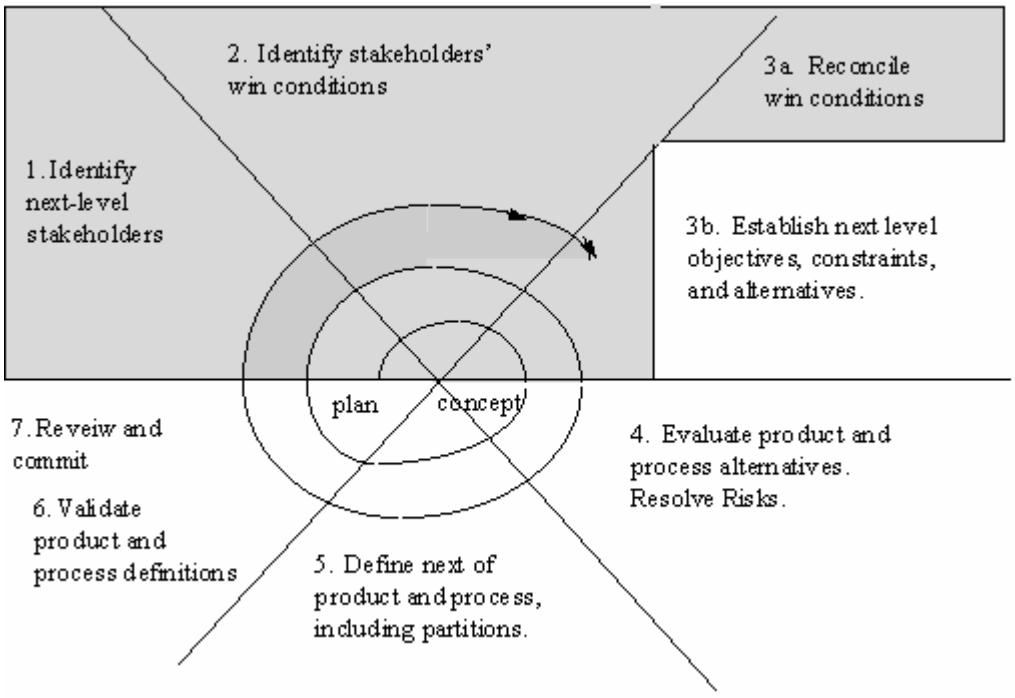



*Figure 2.12 – Win-Win Spiral Model*

Thus, it is evident that the Win-Win Spiral model includes elicitation, analysis (conflict resolution, risk analysis), verification, and management phases of the requirements generation process. An advantage of this model is its well-explained elicitation and analysis phases, which emphasize on the importance of customer participation. However, the model fails to address the requirements specification phase. Moreover, there is no description of intermediate requirements documents generated during the requirements engineering process.

**Summary:** Our objective of analyzing the requirements generation models is to evaluate the features of these models and highlight the positive features, which can be included in the model proposed by our research.

The Requirements Generation Model (RGM) is the only process model that explicitly identifies all five generic requirements engineering phases. In addition, it decomposes the requirements capturing phase and provides explicit guidelines and protocols for the elicitation phase. However, the model still lacks an adequate level of decomposition to represent flow of requirements documents.

Requirements Triage focuses on a single phase and succeeds in providing an elaborate set of activities. The process of assessing the features and determining their impact on the overall development effort presents a very practical approach to requirements engineering. Evaluating the requirements from different perspectives is a critical element and should be incorporated in the requirements generation process. The disadvantage of this model is its lack of coverage of the overall requirements engineering process.

The Knowledge-Level Process Model advocates a scenario-based approach to requirements generation and provides clear elicitation guidelines. Despite identifying the transformation of scenarios and requirements into formal specifications, the model lacks the detailed description about the format and content of these documents. Different levels of abstraction are provided to support a better comprehension of the activities. However, this model is complex and is closely associated with scenarios, and thereby making it difficult to modify.

The SCRAM model presents scenario-based requirements elicitation in an iterative manner. The model provides guidelines for elicitation, verification, and validation activities in the general requirements engineering process. However, no explicit description of the specification, analysis, and management phases are provided. The model neither addresses the issue of transformation of documents nor the representation of the requirements. In addition, activities identified in the SCRAM need to be further decomposed to establish a more uniform and clear understanding of the activities. Regarding documents, SCRAM identifies informal and formal representations of scenarios. However, the representations of the requirements are not explained in sufficient detail.

Finally, the Win-Win Spiral model, based on an iterative approach, provides guidelines for elicitation and analysis. The original Spiral model, proposed by Boehm [Boehm 1988], emphasizes the importance of risk analysis in the overall development process. The Win-Win Spiral model applies these principles to the requirements engineering process. Similar to the triage and knowledge level process model, this model fails to address all the phases of the requirements engineering process. In addition, the Win-Win

Spiral model is somewhat more abstract when compared to the other requirements engineering models.

In general, the requirements generation models fail to cover the entire requirements engineering process, and instead only focus on certain part(s) of the process. In addition, the models lack detailed information about the documentation of requirements. Although some of the models identify intermediate requirements documents, they do not provide clear guidelines on the content and format of those documents. Inadequate decomposition of the phases and activities, and unclear requirements transformation are other problems inherent to the several of requirement engineering models presented in this section.

## 2.4. Research on Recording Requirements

The requirements generation process is not considered to be complete without effective recording and representation of requirements captured during the requirements engineering process. In the past, considerable research have been undertaken to identify content and format of the requirements documents in various requirements generation activities. However, most of the research focuses on addressing requirements documents within certain portions of the requirements generation process rather than covering the entire process.

This section contains a survey of literature on recording and managing requirements documents[5]. In addition, we also discuss available commercial requirements management tools.

### *2.4.1. Requirements Documents*

The current research on requirements documents is limited, with some attempts being made to identify the requirements documents and their characteristics for the individual activities within the requirements generation process. Furthermore, even though requirements generation process flow and related techniques have been proposed in the literature, little research has been conducted to identify and integrate the intermediate documents created during the requirements engineering process

[5] The detailed discussion of requirements documents is presented in Chapter 4.

This thesis is concerned with documents directly affecting the requirements. Documents related to the development process such as the work contract or project schedule are beyond the scope of our research.

#### 2.4.1.1. Requirements Elicitation Documents

Requirements elicitation is an iterative process of discovering and articulating the user needs by communicating with the stakeholders, who include customers, system users, developers, and other affected parties [Sawyer 1997]. Eliciting requirements for the new system has been considered as the initial activity of the requirements generation process. However, in recent years, the requirements engineering community has been emphasizing the importance of understanding the *problem domain* before the elicitation of requirements [Lauesen 2002] [Davis 1993A].

Lauesen identifies that the following intermediate documents need to be produced before eliciting requirements [Lauesen 2002]:

- Description of the present work domain – business activities in the application domain.
- A list of problems in the domain – identifying existing problems in the application domain.
- A list of business goals and critical issues – high level reasons for developing a new system. (For more information on goals, refer to [Cooper 1996].)
- Ideas for the large scale structure of the future system – a vision of how people will function when the software product is used in its intended environment.

According to Leffingwell, requirements engineering starts with an activity, “Problem Analysis”, to understand real-world problems and user needs [Leffingwell 2000]. He advocates that five requirements documents be generated during problem analysis:

- Problem statement – describes the existing problems in the application domain and lists the potential impact of the problem and the benefits of addressing them.
- Root cause of the problem – uncovers the underlying cause of the problem. Root cause analysis can have graphical formats such as the fishbone diagram.

- Stakeholder profile – identifies stakeholders affected by the problem and creates a summary of stakeholders' information needed for the requirements generation.
- Constraints – specifies restrictions on the degree of freedom we have in providing a solution.
- System boundary – identifies the border between the solution and the real world that surrounds the system. A context diagram is a popular form for representing system boundary. The diagram depicts the software system and its surroundings by showing a set of data flows that cross into and out of the system [DeMarco 1978].

Once the problem domain is analyzed and understood, the next step is capturing the user requirements through an iterative process. The elicitation process includes a variety of techniques, such as interviews with stakeholders, questionnaires, and structured elicitation meetings. Special checklist forms can be used to record user needs and requirements elicited by questionnaires and interviews. Sutcliffe recommend the use of scenarios to effectively capture the requirements [Sutcliffe 2002]. Scenarios occur in different forms, from the real world stories to requirements models and specifications. Rolland provides a detailed discussion of what distinguishes the knowledge content contained within a scenario, how a scenario is represented, and how it can be changed or manipulated. Scenario content and format can vary from rich narrative descriptions of a system's use [Kyng 1995] to descriptions of event sequences in tabular formats [Potts 1994] and to the more formal models of system behavior [Heymans 1998].

Ideas and needs elicited from the stakeholders have to be analyzed and modeled to provide a better understanding of the system for the requirements engineer and the stakeholders. In addition, models also enable the requirements engineer to refer to them later for analysis and for new ideas.

#### 2.4.1.2. Requirements Analysis Documents

Requirements Analysis is the process of analyzing user need to arrive at a definition of software requirements and ensuring that a correct software product is being built

[Brackett 1990]. Based on the requirements captured during the elicitation phase, the requirements engineer evaluates the requirements from various standpoints.

Many forms of requirements are available for requirements analysis. Exploring different representations reveals insights that no single view can provide [Davis 1993].

Requirements analysis representations can be divided into several categories – functions, data, and nonfunctional requirements [Lauesen 2002].

**Functions/Features of the System**

- Dataflow diagram - A bubble diagram showing functions and data flowing between the functions. The dataflow diagram provides a compact specification of the data and outlines user tasks in a graphical way. However, it is not suitable for describing user tasks with many variations.
- Viewpoint form – A requirements representation based on the concept of viewpoints by Kotonya and Sommerville [Kotonya 1996].
- Feature requirements – Statements of the functionalities saying "the product shall record/show/compute…"
- Scenario – A case story illustrating user tasks, or a specific case to be tested. Besides explaining the regular user task, scenarios can be structured to include the precondition of the task and possible exceptions to the normal scenario.
- Decision table – Lists all the possible conditions and appropriate actions in a tabular format.
- Decision tree – Provides a tree structure to lay out options and investigate the possible outcomes of choosing those options.
- State diagram (Finite state machine) – Depicts how a certain entity changes its states as a result of various events.
- Object-oriented requirements analysis formats – The conventional requirements generation approach (functional paradigm) models the system as a group of functionalities in an "input-process-output" format [Constantine 1989]. Another

approach is the object-oriented paradigm, which views the system as a collection of objects interacting with each other. Object-oriented requirements analysis forms include use cases and graphical notations such as activity, class, collaboration, and sequence diagrams [Jacobson 1999]. This thesis focuses on improving the quality of the functional paradigm. Hence, the object oriented requirements6 documents are not covered in our research.

**Data Requirements**

It is important to specify the data to be stored in the system whether data is stored in a database or in some other format. Data requirements can be modeled as:

- Data dictionary – A textual data description structured systematically.
- Data model (Entity Relationship diagram) – A block diagram describing data stored in the system and the relationships between the data [Hull 2002].
- Data expressions – Compact formulas that describe data sequences.

**Non-functional Requirements**

The literature includes a few approaches that attempt to represent non-functional requirements[7] [Cysneiros 2002]. These approaches are based on:

- Quality factors – Lists non-functional requirements (quality factors) and uses them as checklist in determining which non-functional requirements need to be addressed for the system.
- Quality grid – Represents non-functional requirements in the form of a grid.
- Goals – This approach treats non-functional requirements as goals that might conflict [Mylopoulos 1992] [Chung 1999].

[6] Refer to [Jacobson 1999] and [Bailin 1997].

[7] Details of nonfunctional requirements are discussed in Chapter 4.

- ER Model – Integrates nonfunctional requirements into data model and characterizes how non-functional requirements affect the data model [Cysneiros 1999].

Besides representing the functional and non-functional requirements, the following information is documented to support the requirements analysis process:

- Priorities of requirements
- Potential risks of requirements
- Cost and effort required for developing requirements.

#### 2.4.1.3. Requirements Specification Documents

The SRS has a major role in the software development process and is used by a wide range of stakeholders. It is challenging to produce a document that is agreeable to all stakeholders and which has a sufficient level of comprehensiveness, preciseness, and completeness. In order to clearly describe all the requirements, the specification is written based on a standardized format with supporting information to aid in the complete understanding for the reader.

The requirements stated in the SRS can be presented in a variety of formats and styles, varying from informal to very formal. The informal approach produces a document written in a natural language so that the relationship between the different requirements is defined based on appropriate groupings. The formal approaches take the same list of requirements and create an equivalent mathematically formal notation of the requirements; this enables establishing correctness and completeness of the elicited requirements. Formal notations require expertise both to produce and understand the document; therefore, natural language is preferred over formal languages even though verification for completeness and correctness is more challenging. Most of the standards and guidelines for writing the SRS assume that it will be written in natural language and often supplemented with diagrams. Standards such as [IEEE 1993] and [DoD 1985] define the structure, quality, and contents of a SRS.

The commonly used approach to represent requirements in the SRS is listing them in a textual form as feature requirements (explained in Section 2.4.1.2). This type of representation is easy to understand and can be referred to by both the customers and developers. However, if the document is not structured appropriately, validation can be difficult since searching for features and checking them may be cumbersome for large specifications.

Representations of non-functional requirements have not been addressed to the extent that functional requirements have. The non-functional requirements, such as reliability, maintainability, and performance, lack formal foundations [Sutcliffe 1998]. Nevertheless, promising work has been done by [Chung 1995].

Although the exact contents of a requirements specification vary from situation to situation, several types of information are included in most requirements specifications. In addition to functional and nonfunctional requirements statements, the SRS contains a variety of technical and non-technical text, such as [Hull 2002]:

- Description of the environment and objectives of the system
- Definition of the scope of the requirements
- Stakeholders description
- Supplementary models used in deriving the requirements, such as DFD and ER diagrams.

Traditionally, requirements have been documented and modified in a software requirements specification. However, in recent years requirements are starting to be maintained in a database rather than a document.

#### 2.4.1.4. Requirements V&V Documents

During requirements V&V, documents created in the preceding activities, i.e. elicitation, analysis, and specification, are checked against various quality characteristics and customer intent.

Verification ensures that the requirements in the SRS adhere to quality attributes and comply with document standards. According to Lauesen, verification can be divided into two examination types – content checking and structure checking [Lauesen 2002].

Content check verifies if the specification is complete, that is includes all the necessary information such as system goals and different types of requirements (data, functional and quality requirements). Consistency check compares different parts of the specification to detect missing parts or inconsistencies. For example, the CRUD (Create, Read, Update, and Delete) matrix checks consistency between data and tasks. Other consistency checks include event check, information needs, and so on [Lauesen 2002]. In addition, a checklist form needs to be created to organize the verification process.

On the other hand, structure check verifies that the "pattern" of the specification complies with the standard format, such as numbering scheme, explanation of diagrams, cross-references, and indexing. Furthermore, to enhance the overall organization, every requirement in the SRS must be labeled. "The convention must be robust enough to withstand additions, deletions, and changes in requirements over time" [Wiegers 2001].

Validation ensures that the requirements satisfy the user needs [Wallace 1997]. Wiegers advocates that all requirements contained in the SRS be traceable to its origin so as to guarantee that all stakeholders know the rationale behind the requirements included in the SRS [Wiegers 2001]. The origin may be a scenario, use case, idea, business rule, government regulation, or some other external source. A requirements traceability matrix can assist in the process of tracing every requirement to its source.

The context diagram provides an overview of the required product interfaces and can be used for verification and validation. An overview of the interfaces provided by the context diagram can serve as a checklist of what is to be developed. In addition, most customers readily understand the context diagram, and can help in spotting incorrect or missing interfaces or system components. Scenarios are also effective for validation [Carroll 1995].

Prototyping is the most widely used approach for the purposes of validating elicited requirements [Pressman 2001]. Prototyping enables the user to gain a better understanding of the requirements and provides important user feedback.

#### 2.4.1.5. Requirements Management Documents

Requirements management is a systematic approach to supporting and monitoring the requirements generation activities. In addition, requirements management also controls the constant changes to requirements during the software development process.

Traceability is helpful in monitoring the relationships among requirements documents. Moreover, traceability is needed to compare requirements against other information. An ideal project should include three types of traceability [Pressman 2001] [Lauesen 2002]:

- Features traceability – Tracing from needs to requirements to ensure that all user needs are reflected in the requirements.
- Source traceability – Tracing from requirements to needs to ensure that all requirements are necessary.
- Dependency traceability – Indicates how requirements are related to one another.

Traceability among requirements is most frequently represented by a traceability matrix in a table format. For example, the goal-domain matrix checks if each business goal is addressed by one or more tasks identified, and in turn, signifies if each task supports some business goal. While the goal-domain matrix only illustrates if relationships exist between the goals and tasks, the QFD (Quality Function Deployment) matrix depicts the value of goals and function costs [Lauesen 2002]. Traceability is often facilitated by the use of requirements management tools, which are discussed in the next section.

During software development, the SRS is modified as new requirements found in the later phases of the development or to reflect corrections to the existing requirements. Therefore, it's important to increase the SRS's version number when changes are made, maintain a list of changes, and keep all requirements in a database so that history of requirements evolution can be accessed easily whenever necessary [Lauesen 2002]. In addition, traceability assists the change management process.

**Summary:** As presented in the preceding sections, the literature identifies several ways of representing requirements. However, these attempts focus on only part(s) of the requirements generation process. In general, the documents proposed address only high level objectives of the requirements engineering process. Furthermore, the transformation

of these documents is not clearly presented; also, how these documents are integrated into the requirements generation process is not well-defined.

To obtain better insights into the representation of requirements, we evaluate several commercial requirements management tools in the next section.

### *2.4.2. Requirements Management Tools*

The requirements engineering process is supported by a number of tools to automate the requirements generation and change management activities. Tools also help in optimizing development costs, delays, and quality. In addition, requirement management tools provide guidance to the requirements engineer by identifying templates for requirements documents. This section discusses five commercial tools – CaliberRM, DOORS, Reconcile, RequisitePro, and RTM Workshop – by evaluating their approaches on how to capture, analyze, and refine requirements while maintaining consistency and traceability of the requirements.

#### 2.4.2.1. CaliberRM

Borland CaliberRM[8] is an enterprise requirements management system (Figure 2.13). It is designed to facilitate collaboration, impact analysis, and communication in the definition and management of changing requirements, helping teams improve product quality and reduce the risk of project failure.

CaliberRM includes a feature for identifying and documenting the links between requirements in an online document accessible to everyone in the organization. In addition, it provides tools for identifying requirements inconsistencies and determining the impacts of requirements changes. Furthermore, this tool includes the standardized document templates and also allows the users to define their own document templates. Standard reports, traceability matrices, and customizable requirement grids can be created and printed.

[8] Refer to http://www.borland.com/caliber/index.html

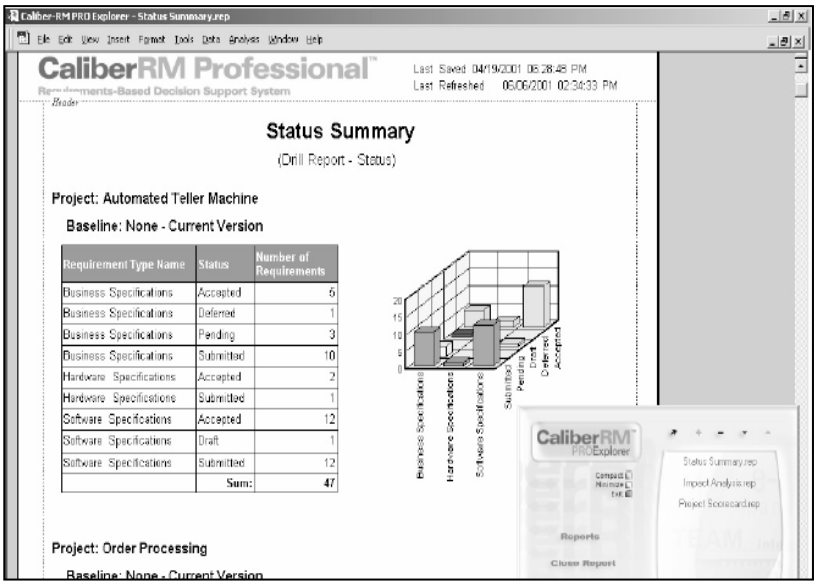



*Figure 2.13 – Snapshot of CaliberRM*

One of the strengths of CaliberRM is its ability to analyze requirements in several languages including English, French, Spanish, and Swedish. It also integrates into different development environments through its source code control interface. However, in spite of its flexibility, CaliberRM does not manage requirement change requests, which are manually processed. The product relies only on the change management capabilities of third-party tools, such as ClearQuest, StarTeam, PVCS Dimensions, through the source code control interface. This makes it unsuitable for projects where changes must be validated before they are baselined, without the need to rely on an external tool.

#### 2.4.2.2. DOORS

The main feature of the DOORS (Dynamic Object Oriented Requirements System) is editing documents "as if they were in a word processor, but with the ability to add attributes, filters, links and perform configuration management in the same window" [Telelogic 2002].

DOORS supports the linking between requirements and other project related information. Traceability of information enables identifying the impact of requirement changes and determining internal inconsistencies among requirements (Figure 2.14). Change

management is supported by a built-in change proposal system that lets users submit requirements change proposals and reviews [Systemsguild 2002]. The impact of changes is identified through the traceability links and tools that show a graphical overview of the requirements.

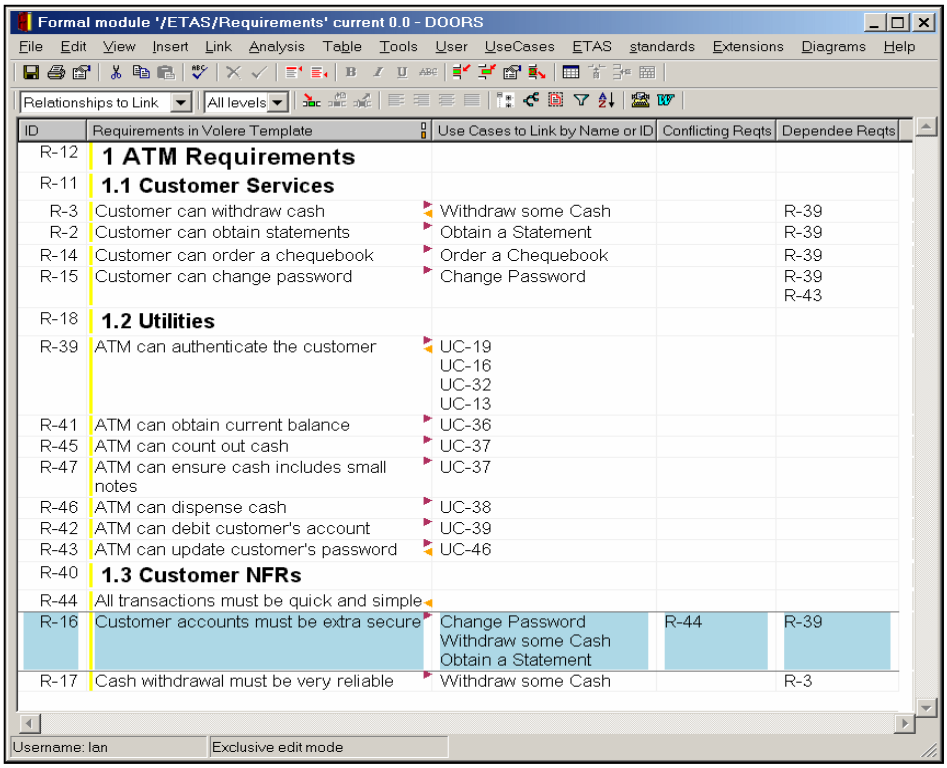



*Figure 2.14 – Snapshot of DOORS*

DOORS facilitates generating requirements documents according to templates proposed by IEEE and INCOSE. These templates are comprehensive and may not be feasible for the documentation of requirements in small projects. Hence DOORS includes the capability of modifying and distributing the templates throughout the organization. Requirements can be prioritized based on the values of defined attributes. The selection of a same priority level provides a view of the related requirements. This helps in defining the project scope based on the priorities associated with the requirements.

DOORS ensures the consistency of the requirements changes through its bidirectional integration into third-party change management tools, such as CM Synergy, ClearQuest, and PVCS Dimensions. In addition, it facilitates the mapping of requirements with design

and testing tools such as AllFusion Component Modeler and Tau UML Suite, TestExpert, and Tplan. A drawback of this tool is that it is not as effective as RTM workshop in maintaining the consistency of requirements of multiple projects.

#### 2.4.2.3. Reconcile

Reconcile[9] is a requirements management tool that uses the Microsoft Word architecture. It requires minimal initial resource investment and has full integration with popular development and testing tools. Reconcile includes a user-friendly interface which minimizes learning curves and switching tasks. This tool also provides pre-defined requirement types, which are easily modified. Furthermore, Reconcile offers a web interface to allow the sharing of any Reconcile reports across the Internet or an intranet, using its own integrated reporting and publishing tools. In addition, Reconcile tracks the source of each requirement, including its priority and status the following functionalities (Figure 2.15). Moreover, issues and defects identified during testing or development are immediately associated with the requirement that is affected, allowing team members to assess impact [Volere 2004].

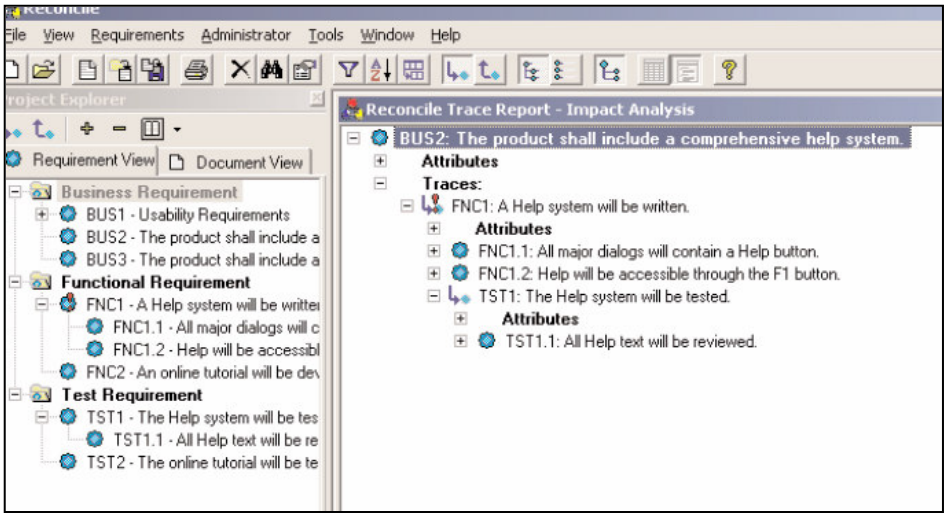



*Figure 2.15 – Snapshot of Reconcile*

An advantage of Reconcile is the bidirectional link between the product's interface (and thus its repository) and the requirements document displayed through the word

[9] Refer to http://www.compuware.com/products/reconcile.htm

processing tool. Clicking on a requirement on the product interface brings the corresponding paragraph in the word processing tool interface. Also changing the hierarchical position of a paragraph in the document is reflected in the Reconcile repository.

Reconcile lacks requirements baseline management due to the lack of maturity of this new-to-the-market product. In addition, the product lacks the capability to associate images or notes with the requirements. Another drawback of Reconcile is that it does not allow a free description or custom attributes to be attached to the requirements. Version management is another aspect that this product fails to address.

#### 2.4.2.4. RequisitePro

The main function of RequisitePro[10] is to link a database of requirements to documents in Microsoft Word. This ensures that requirements can be handled in a natural way through documents.

RequisitePro provides a suite of tools which helps in ensuring the traceability of requirements. In addition, this tool includes a set of functions for determining the impact of requirement changes. Furthermore, analysis activities are supported by requirement attributes, which can be used for sorting, filtering, and searching of requirements (Figure 2.16).

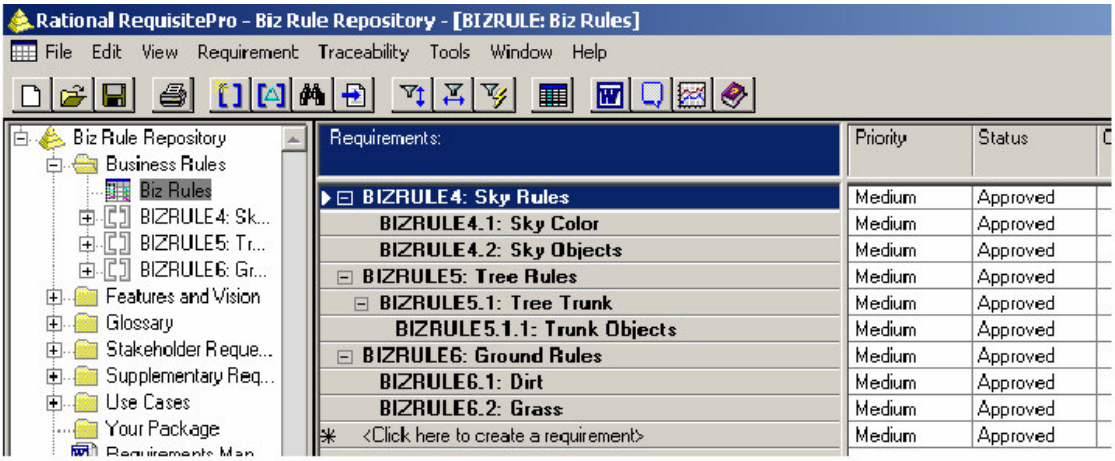



*Figure 2.16 – Snapshot of RequisitePro*

[10] Refer to http://www-306.ibm.com/software/awdtools/reqpro/

Change management is facilitated through the tracing functionality. In addition, all changes in the requirements are recorded for future reference and analysis. The tool produces documentation through templates which conform to different document standards. As in DOORS, RequisitePro enables the users to change the formats to suit their organization and needs.

An advantage of RequisitePro is its web interface, RequisiteWeb. This interface provides the same features as the Windows interface, except that the requirements cannot be moved within the classification tree. The Web interface can be customized by editing a CSS (Cascading Style Sheets) file. This makes the product suitable for distributed projects.

RequisitePro does not specifically warn of the impact of a change on linked requirements. Even though it informs users about the occurrence of changes, the analysis of the impact due to the changes must be done manually. Another weakness of RequisitePro is its inability to append to the requirements document once the requirements document has created through the tool interface [Yphise 2002]. The update can only be done manually by cutting and pasting the addition. On the other hand, the product maintains a bidirectional and "live" relationship between the repository and the requirements document when existing requirements are updated. This makes the product unsuitable for projects where numerous requirements can be added directly through the tool interface.

#### 2.4.2.5. RTM Workshop

RTM Workshop[11] is designed to facilitate and streamline the enterprise engineering process[12]. It uses word processing packages of companies such as Adobe, Interleaf, and Microsoft for its user interface. As users identify requirements in the word processor, the requirements are automatically saved into the icCONCEPT database (Figure 2.17). Users continue working in their preferred word processing environment to refine the initial requirements into the final specification.

[11] Refer to http://www.nrt.se/nrt/krav/Produkt/RTM-Workshop.pdf

[12] Enterprise engineering is defined as the process of transforming an idea from "concept to delivery".

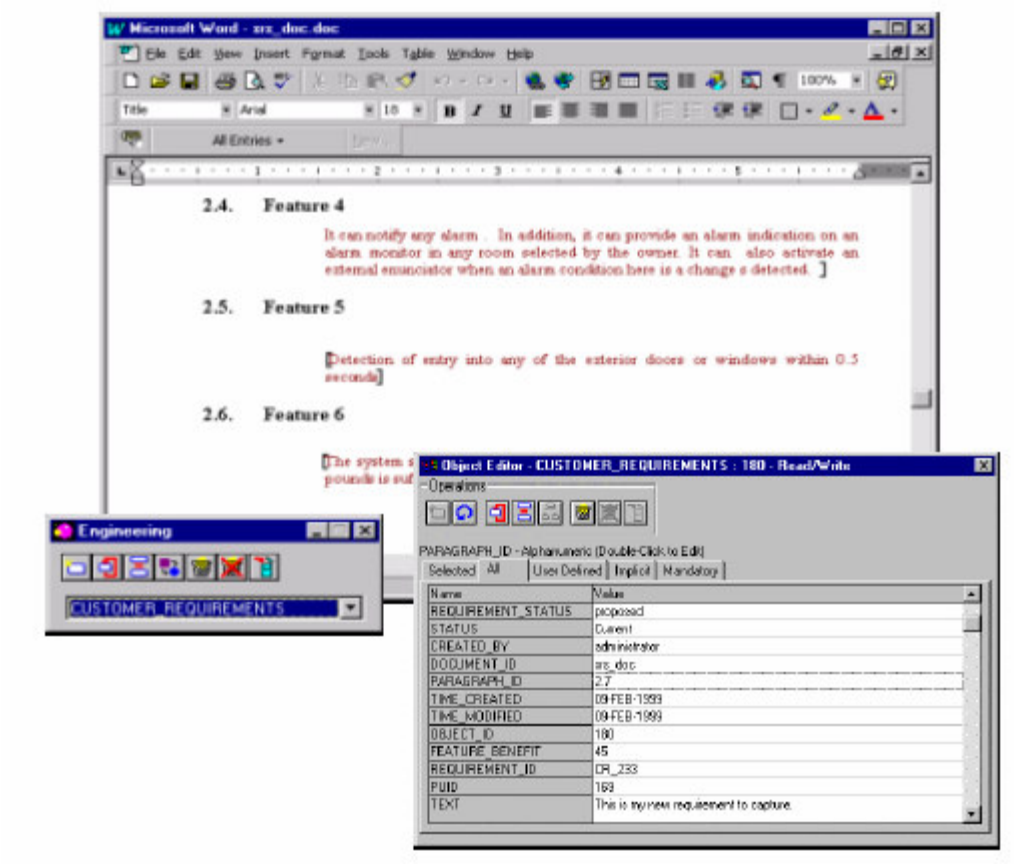



*Figure 2.17 – Snapshot of RTM Workshop*

A key project management concern is determining how to manage the volumes of interdependent information provided by different tools while simultaneously ensuring that the requirements are being satisfied. RTM Workshop offers an interface into leading software development tools and traces the project requirements to the user needs to guarantee that all requirements are necessary.

The product facilitates the submission of requirements changes and integrates them into a validation workflow; this ensures that the changes are taken into account. In addition, the stakeholders are informed of changes to the requirements whenever they occur. When change proposals occur in an RTM Workshop managed project, all affected items are graphically highlighted. The user can then assess the magnitude of the change and conduct a full impact analysis. In addition, RTM Workshop maintains a complete change history, including records of the impacted requirements and the rationale for each change.

RTM Workshop is suitable for projects where a large number of users are involved and different access privileges are required. An added benefit is the simple and easy to use interface of RTM Workshop. Furthermore, this tool supports distributing and synchronizing the requirements of projects among various repositories. The drawback of this tool is that the document sets cannot be compared for consistency or similarities. Another disadvantage is that this tool cannot format the documents that are to be published. Hence, this tool has to rely on other tools for publishing of the requirements documents.

**Summary:** The requirements management tools guide the requirements engineer through the requirements capturing and analyzing activities by providing methods and templates. In addition, the tools ensure consistent development in spite of constant requirements changes.

However, the tools have a limited number of requirements documents, which mainly list the requirements. Graphical representations of requirements can be imported from other sources, but the templates and formats to create requirements in a graphical representation are few. As a consequence, the task of analyzing requirements becomes difficult.

## 2.5. Summary of Research Issues

In this chapter, we present the related work for our research. We discuss the existing SDLC and requirements engineering models, highlighting their insufficiencies at representing intermediate requirements documents. Based on the analysis of different models, we propose a refinement of the RGM model with a focus on the recording and usage of requirements documents throughout the requirements generation process. This section is a prelude to Chapters 3 and 4, which present the proposed refinement of the RGM and the requirements document evolution respectively.

The literature review shows that requirement generation models lack an adequate level of decomposition. In addition, the models do not provide comprehensive guidance on the characteristics of the requirements documents generated during the requirements

engineering process. Furthermore, the current requirements engineering models fail to clearly specify the relationship of the documents and related activities.

This research has two main objectives:

- Identify requirements documents that support the objectives of activities
- Determine content and format of the intermediate documents as requirements evolve during the requirements engineering process.

In brief, the synchronization of the documents with the related activities results in a seamless transformation of requirements ensuring correctness and a minimal loss of information.

We propose a comprehensive requirements generation model spanning the entire requirements engineering process. The model is composed of two main parts, (1) a framework of requirements engineering activities at a sufficient level of abstraction and including information about the activity objectives, participants, and strategies on the process steps, and (2) guidelines and templates of the content and format of various requirements representations used throughout the requirements engineering process.

In order to develop the proposed model, following issues need to be considered:

- **Identifying the requirements model and the activities:** For this research, it is necessary to identify a model that addresses all the major requirements engineering phases and, at the same time, is easily changeable so that the flow of requirements can be clearly represented. We choose the RGM model after the analysis of several models because it possesses the qualities essential for this research. However, because the RGM has a high level of abstraction it is necessary to refine the model so that we can represent the seamless flow of information through documents.
- **Analyzing activities and determining their objectives:** The current requirement engineering models do not clearly define the objectives of associated activities. Therefore, it is crucial to determine the real objectives of the activities once the model is decomposed and refined. Once the activities are identified and the

objectives are determined, the requirements model has the necessary abstraction and organization to illustrate the flow of information.

- **Identifying document characteristics:** The documents created in the requirements generation process play a critical role in the generation of a well-defined SRS. Hence, it is necessary to identify the content and format of these documents. This entails analyzing documents proposed by both researchers as well as those used in the industry.
- **Synchronizing documents and activity objectives:** The current requirements engineering literature does not provide adequate guidance in the creation of documents during the requirements generation process. The content and format of both the input and output documents need to be directed by the objectives of the activity. Hence, it is necessary to synchronize the requirements documents with the related requirements engineering activities so that the objectives of these activities are satisfied through the information presented in the documents.
- **Determining the evolution of requirements:** A problem that the requirements engineers face is in implementing a seamless transformation of requirements as they pass from one activity to another. In order to simplify this task of the requirements engineer, it is essential to trace how the content and format of the documents evolve in the requirements generation process, such that there is a minimal loss of information. The tracing of information ensures the correctness of requirements as requirements are captured, analyzed, documented and verified. By analyzing the requirements documents and flow of related activities, we identify the evolution process of requirements as documents transform through the requirements engineering process.

In the next chapter, we discuss the decomposition of the RGM model to facilitate the identification of documents and to provide a clear understanding of the requirements generation process.

# CHAPTER 3. THE REFINED REQUIREMENTS GENERATION MODEL

## 3. The Refined Requirements Generation Model

As we described in Chapter 2, the current requirements generation models fail to articulate the importance of the intermediate representations of requirements, and do not emphasize the evolution of the requirements throughout the requirements engineering process. To overcome this problem, we introduce the refined Requirements Generation Model in this chapter.

The model represents a structured process consisting of well-defined activities and objectives. Not only are the sequences of activities illustrated in the model, but also the input and output artifacts (requirements documents) are identified based on the *objectives* of related activities. In addition, the documents from the different activities are integrated so that there is a seamless flow of requirements transformation throughout the entire requirements engineering process.

The model consists of two parts – (1) decomposed and refined activities and their objectives, and (2) detailed characteristics of requirements documents generated throughout the requirements engineering process. Chapter 3 describes the refined model and the decomposition of its activities to represent the flow of requirements, whereas Chapter 4 presents the characteristics of intermediate and final representations of requirements, starting with the problem identification phase and extending through the generation of the software requirements specification (SRS).

This chapter begins by presenting an overview of the model; the subsequent sections explain decomposed activities and their objectives in detail.

### *Overview of the Refined Requirements Generation Model*

The refined RGM model consists of six major phases – Conceptual Overview, Problem Synthesis, Requirements Capturing, Global Analysis, Requirements Specification, and Requirements Verification and Validation (Figure 3.1). The RGM model has a

"waterfall" characteristic to focus on requirements generation as a distinct process at the start of software development [Sud 2003]. The refinement of the RGM model is based on the "separation of concerns" [Aksit 2001], which emphasizes the organization and decomposition of the complex process into simpler activities by addressing small sets of concerns. Each of the phases in the RGM model, illustrated in Figure 3.1, is further decomposed into smaller activities with well-defined objectives. The first two phases represent the problem domain, which gathers information about the existing problems and needs. The remaining phases correspond to the solution domain; they capture the requirements of the solution.

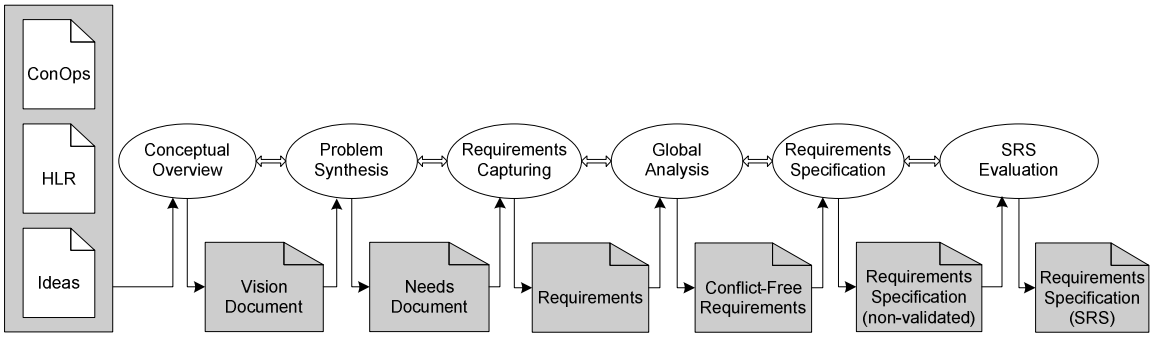



*Figure 3.1 – Overview of the Refined RGM Model*

In the following sections, each activity in the RGM model is described in detail. The description includes the primary objective, rationale of the activity, participants involved and steps required to complete the activity. In addition, input and output documents are identified based on the objective of the activity.

## 3.1. Conceptual Overview

"The product concept provides a trigger for the requirements engineering process to begin. That trigger might be an improvement in customer service, a future need, improvement of existing system, or need to use some available technology" [Macaulay 1996]. The starting point of the requirements generation is capturing the overview of the customer's viewpoint concerning the problem and the proposed system. Conceptual Overview attempts to recognize the need for the new system from the business and

operational perspectives, and ensures a common understanding of project objectives among the stakeholders. The product objectives and vision are the basis of what the system requirements should represent. Therefore, to generate a common product vision, a list of inadequacies in the current system is generated, and the benefits of the new system are justified through customer meetings. To aid this process, various input documents, such as High-Level Requirements (HLR), Concept of Operations (ConOps) or ideas, can be provided by the customer. A brief description of these inputs is given below:

- **HLR** – When a software system is a part of a larger system, HLR is provided by the system engineering process, which is conducted prior to software development. HLR includes high-level functional requirements for the software and its interface to other parts of the system.
- **ConOps** – is produced based on customer/user analysis of the current system. It describes the proposed system characteristics in a user domain language. The major components of the ConOps include the description of the current system, justification for changes, concept of operations, and benefits of the proposed system [Fairley 1996].
- **Ideas** – Often times customers/users have their own perception of both the problem and the solution. They articulate their ideas in application domain terms. Requirements engineers must capture those ideas and document them.

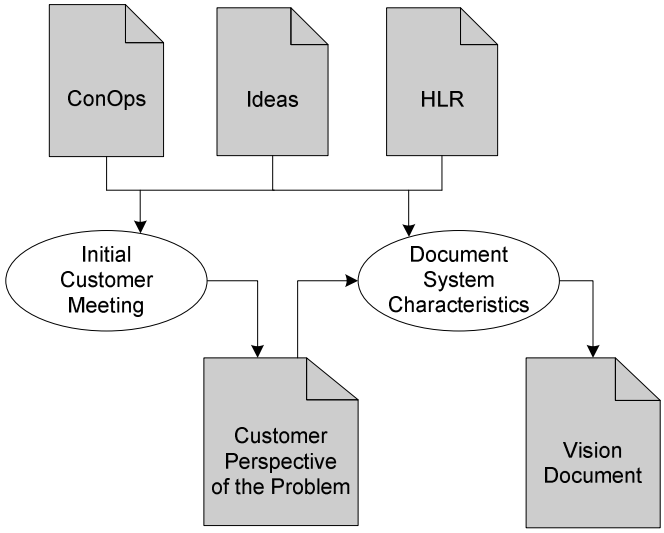



*Figure 3.2 – Conceptual Overview*

Figure 3.2 depicts the Conceptual Overview process and input/output artifacts. The main process consists of conducting initial meetings with customers and generating the Vision document.

### *3.1.1. Initial Customer Meeting*

As illustrated in Figure 3.2, the requirements engineering process begins by meeting with the customers, who initiated the project, to develop an understanding of the project goals and the benefits of developing/upgrading a new system. The main objective of the initial interaction is to understand and record the customer's perceptions of the problem and the solution, and the rationale behind it. These meetings are vital to the project as they not only help the requirements engineer to get the initial idea of the rationale and goal of the project, but also assist the customers in identifying the success factors of the project. Setting the goals and success factors of the project helps to evaluate alternative solutions and to make appropriate decisions in later phases of the requirements engineering process.

The most common techniques of the interaction are meetings or interviews. During the meetings, the requirements engineer attempts to capture the customer's view point on the problems faced in the current system and possible solutions to overcome it, and the benefits of the solutions to current business operations.

The level of interaction with the customers depends on the available documents. If the project is initiated based on system engineering results, a high-level solution is already determined. Therefore, initial customer interaction is targeted more at understanding and clarifying the predefined objectives and rationale of the proposed system, rather than identifying them. In the case of the ConOps document, customer analysis also provides an initial statement of the problem and solution. These are, however, from the user's perspective and written in an application domain vernacular. Hence, customer interaction is required to establish a common understanding of the ConOps and to expand on the rationale and benefits described in the ConOps. When the input is only ideas from the customer, more interaction is needed to identify the proposed system characteristics, such

as current problem, relevant high-level solutions, and impact on the current business operations.

The guidelines for documenting the proceedings of the meetings are presented in Section 4.1.1.4. The output of this activity, the Customer's Perspective document, assists in generating the Vision document in the subsequent activity.

### *3.1.2. Document System Characteristics*

After the initial interaction with the customers, the requirements engineer should document the meeting notes to generate the Vision document. The objective of the documentation process is to ensure that both the customer and the requirements engineer have clear and unified goals for the proposed system. The Vision document is an important reference which gives high-level understanding for conducting subsequent activities in Problem Synthesis. Moreover, this document can be used as a guide in negotiating requirements because it includes the goals to be achieved for the proposed system to be considered successful.

The Vision document should provide a proposed system overview and highlight the problems of the existing system. In addition, the Vision document justifies the rationale of the proposed system by listing the benefits of that system.

Based on the artifacts used as input, the documentation process differs. In case that the HLR or ConOps is produced before the requirements engineering process starts, the vision is well documented. Therefore, the appropriate sections in the HLR or ConOps can be modified to establish the Vision document according to the Initial Customer Meeting. If no document is available, the Vision document is created in accordance with the output of the Initial Customer Meeting and should follow the format given in Appendix B.1.4. Detailed description of the Vision document is illustrated in Section 4.1.1.5.

When the Vision document is generated, it should be given to the customer for review. In general, customer review is a very important part in the requirements engineering process because the main objective of the requirements engineering is to ensure creating a description of a system which reflects the customer needs.

## 3.2. Problem Synthesis

When the high-level description of the problem, possible solution, and its rationale is identified in Conceptual Overview, the next set of activities focus on a comprehensive analysis of the current problems and the customer needs to overcome those problems. Problem Synthesis has two major parts, Problem Analysis and Needs Generation (Figure 3.3).

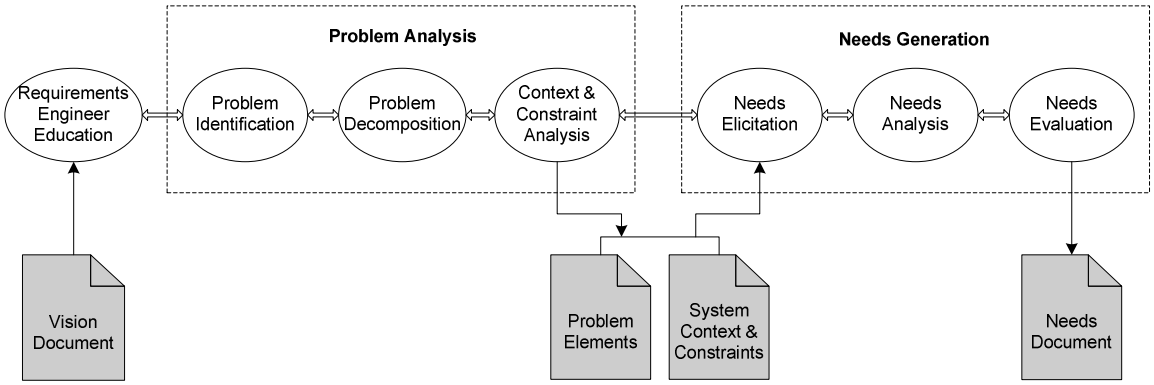



*Figure 3.3 – Overview of Problem Synthesis*

The main objective of Problem Analysis is to identify the real problems in the current practice and to determine the root cause(s) behind those problems. Problem Analysis is important because without understanding the problem, it is difficult to obtain the "right" solution. After developing an understanding the problems and the reasons behind them, the user needs are then generated. The objective of Needs Generation is to express a solution to the problems from the perspective of customer needs. Needs are also useful in requirements validation because the requirements have to reflect and trace back to the user needs.

The Problem Synthesis process begins by identifying the high-level problems, and then analyzing the root causes. Based on the root causes, complex problems are decomposed into a set of smaller, distinct problem elements. In addition to analyzing the problems, the requirements engineer also determines the system boundary and constraints. Once the problem elements are identified, user needs are generated by analyzing each problem

element within the confines of the system context and constraints. Needs are then prioritized by the stakeholders. The final output of the Problem Synthesis phase is the Needs document. A detailed explanation of each activity in the Problem Synthesis is given below.

### *3.2.1. Requirements Engineer Education*

Before analyzing the problem in depth, it is necessary that the requirements engineer gains an understanding of the application domain (problem domain). Application domain knowledge is concerned with the information pertaining to the general area where the system is applied. To understand requirements for a proposed system, one must have background information about the operations of the existing system [Kotonya 1998]. Therefore, the customer should educate the requirements engineer about the application domain concepts and relevant artifacts. The main objective is to provide an overview of the application domain and the details of parts relevant to the project. This is an important activity because the requirements engineer has the responsibility to analyze the problem *within* the application domain and then to elicit user needs [Arthur 1999].

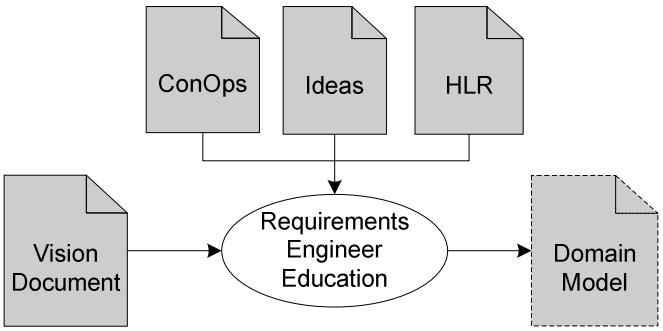



*Figure 3.4 – Requirements Engineer Education*

In the education activity (Figure 3.4), the customer should assume control and conduct educational activities, such as presentation and meetings, interviews, or demonstrating the actual process at the workplace. Documents describing the application domain and its functions, such as ConOps, HLR, and/or other artifacts can assist the requirements engineer's learning process. Based on the things learned during this activity, the

requirements engineer can create a formal document characterizing the application domain, which can be used for the downstream Problem Analysis activities. However, the creation of the document is not mandatory. If Requirements Engineer Education is omitted, the requirements engineer must develop on understanding of the application domain on his/her own time during Problem Analysis.

### *3.2.2. Problem Analysis*

It is important to have an adequate understanding of the problem before attempting to develop a solution. Unfortunately, the importance of this activity has been realized only in recent years [Davis 1993] [Leffingwell 2000]. The main objective of Problem Analysis is to identify the problems and underlying causes. The list of problems identified during Problem Analysis can serve as validation criteria for the user needs and requirements, and help ensure that the proposed solution addresses the actual problems.

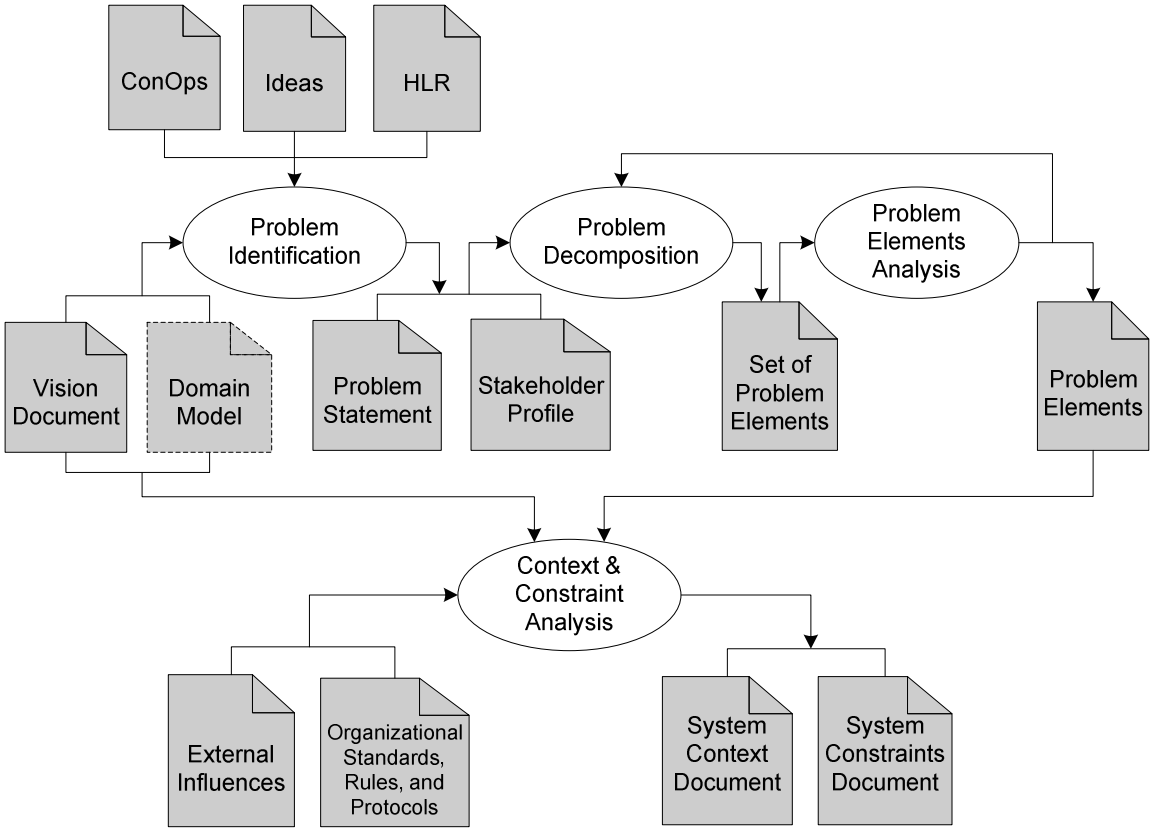



*Figure 3.5 – Problem Analysis*

As illustrated in Figure 3.5, Problem Analysis consists of four activities:

- **Problem Identification** – the real problem and the possible causes for the problem are identified.
- **Problem Decomposition** – the problem is decomposed into a set of smaller and distinct problem elements.
- **Problem Elements Analysis** – the problem elements are organized and evaluated for completeness.
- **Context and Constraints Analysis** – the boundaries and limitations of the system are determined.

Each of these activities is explained in the following sections.

#### 3.2.2.1. Problem Identification

The main objective of this activity is investigating and identifying/substantiating the real problem. By understanding the real problem, the task of determining the user needs becomes easier. While identifying problems, the requirements engineer also needs to identify stakeholders affected by these problems.

The steps in this activity differ depending on the input source. If the input is the HLR or ConOps, the requirements engineer can assume that the problem is identified correctly. Subsequently, the Problem Identification process focuses on understanding and validating the problems. However, if the input is user ideas, the process is focused on investigating, verifying, and identifying the problems. Techniques such as interviews, brainstorming, and ethnographical studies can be used for this activity.

Next, stakeholders affected by the problems are identified, and detailed information about the stakeholders, such as responsibilities and application domain knowledge, is recorded in the Stakeholder Profile document (Figure 3.6). The information about the stakeholders is an essential reference for further communication and clarifications. Based on observation of in the work place and interviews with the stakeholders, the Problem Statement is created, and describes the problem, affected stakeholders, and the benefits of resolving the problem.

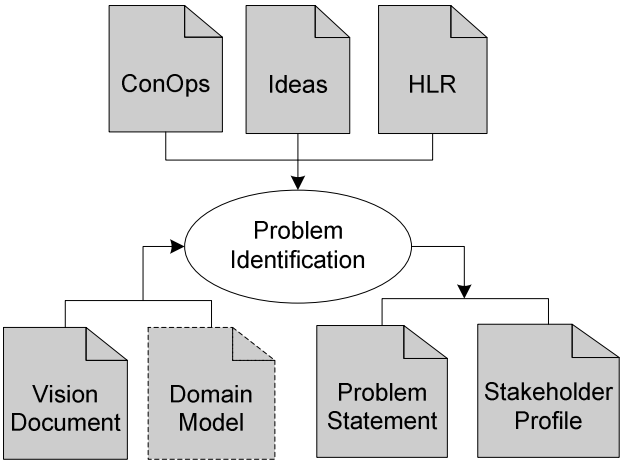



*Figure 3.6 – Problem Identification*

#### 3.2.2.2. Problem Decomposition

This activity attempts to partition the problem into sub-components (problem elements). This decomposition facilitates easier handling of the complex problems in the subsequent activities. In order to achieve the decomposition objective, it is imperative to identify the possible causes for the principal problem. Once the root cause is identified, the problem is broken down into its constituent problem elements (Figure 3.7). This activity provides insights into the problem and helps evolve a better understanding of the user needs.

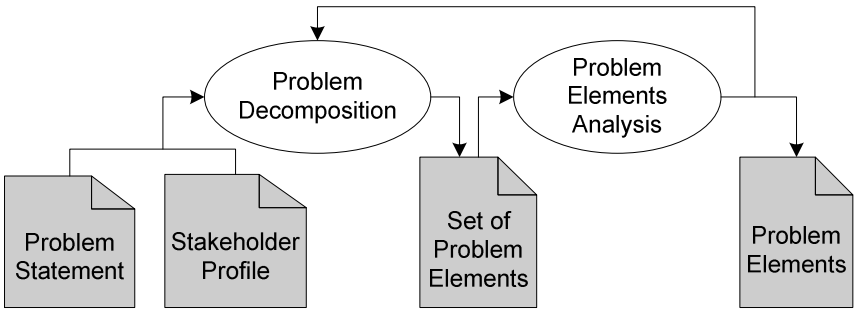



*Figure 3.7 – Problem Decomposition and Analysis activities*

The first step in this activity is to determine the cause for the overall problem through root cause analysis approach. Root cause analysis is a systematic approach to uncovering the underlying cause for a problem or symptom of a problem [Leffingwell 2000]. The

fishbone diagram is a common technique associated with this approach. Potential causes for the problem are analyzed, and the root cause is identified in consultation with the customer. Once the cause(s) of the problem is determined, the sub-problems are identified through the "divide and conquer" approach.

The output of this activity is a list of problem elements; this is critical for the project since the user needs are generated based on this list. The Set of Problem Elements document is then passed to the next activity for organization and evaluation.

#### 3.2.2.3. Problem Elements Analysis

The objective of this activity is to structure and analyze the problem elements. At the end of this activity, the requirements engineer decides whether further decomposition is needed (Figure 3.7). The output of this phase is a set of problem elements that is complete and structured.

Organization of the problem elements is often based on their importance/priorities, which is assigned by the customer. Priorities can be determined using the analytic hierarchy process technique [Salo 1997], which requires every problem element to be compared to each other. Thus, the priorities are assigned relative to each other. This method is useful if the number of problem elements is small. However, if the number of problem elements is large, several other classification schemes such as related features, schedule importance, and so forth can be used.

Once the problem elements are prioritized, they are analyzed for clarity to prevent misinterpretations. In addition, the requirements engineer checks the problem elements for completeness. If any problem element needs to be further decomposed, the decomposition activity (Section 3.2.2.2) is repeated.

#### 3.2.2.4. Context and Constraint Analysis

The objective of this activity is to determine the system boundaries and the constraints that must be met. This activity is crucial because it limits the scope of the system and helps to define non-functional requirements. The organizational standards, rules,

protocols document, and the external influences document are used in determining the scope and constraints of the system (Figure 3.8).

Context Analysis specifies the boundary which separates the proposed system from its environment. Identifying the boundary of a system helps in determining the inputs to the system and corresponding outputs. Interviews are commonly used for this activity. The following questions can be used to obtain the system information [Leffingwell 2000]:

- Who will operate the system?
- Who will perform system maintenance?
- What data is generated by the system?

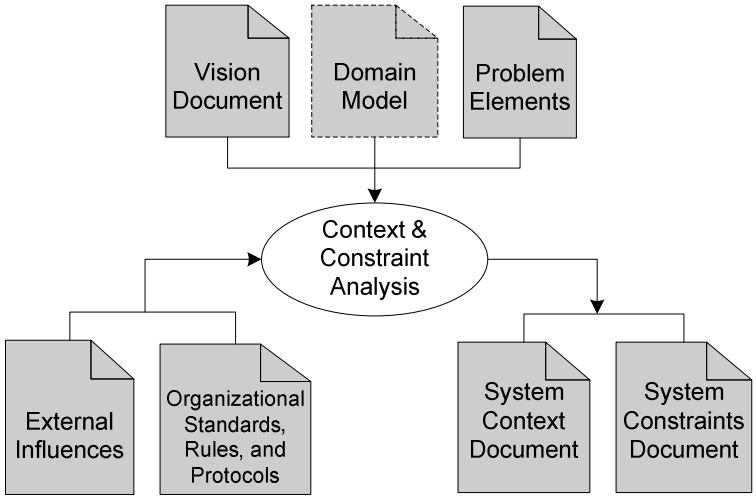



*Figure 3.8 – Context and Constraint Analysis*

The System Context document includes information about the actors interacting with the system and data exchanged between the proposed and external systems.

In addition to determining the system context, the requirements engineer performs constraints analysis, which identifies the restrictions imposed on the system. Potential categories of constraints are listed in Table 3.1 [Leffingwell 2000].

The output of constraints analysis is the System Constraints document. This document, along with the System Context document and Problem Elements, are then passed to the

Needs Generation phase for gathering the user needs. System constraints like security, performance, etc., are used to derive non-functional requirements.

| Constraint Type | Considerations |
|---|---|
| Economic | Is there any procurement costs involved with the hardware/software to be deployed, if any? |
| Technical | Is there any technology that should be explicitly avoided during development? |
| Political | Are there any persons specifically affected by the system, apart from the designated stakeholders? |

*Table 3.1 – Constraint Categories*

### *3.2.3. Needs Generation*

After determining the problems in the current system and analyzing the root causes of those problems, the needs generation phase is entered. This phase focuses on the transformation of the problem elements into a set of user needs. The stakeholders play an important role in the transformation process. Needs represent the customer's perspective of the solution to their problem. Derived from the problem elements, the needs are expressed in an informal manner using terms germane to the application domain.

On completion of this phase, a well-defined problem domain characterization is specified through the Problem Statement, System Context and Constraints document, and the Needs document.

As depicted in Figure 3.9, the Needs Generation phase begins with the elicitation of needs, and is followed by an analysis of those needs. Conflicts among needs are then identified and resolved, and are then validated by the stakeholders. The activities in this phase are iterative in nature, and are discussed in the following sections.

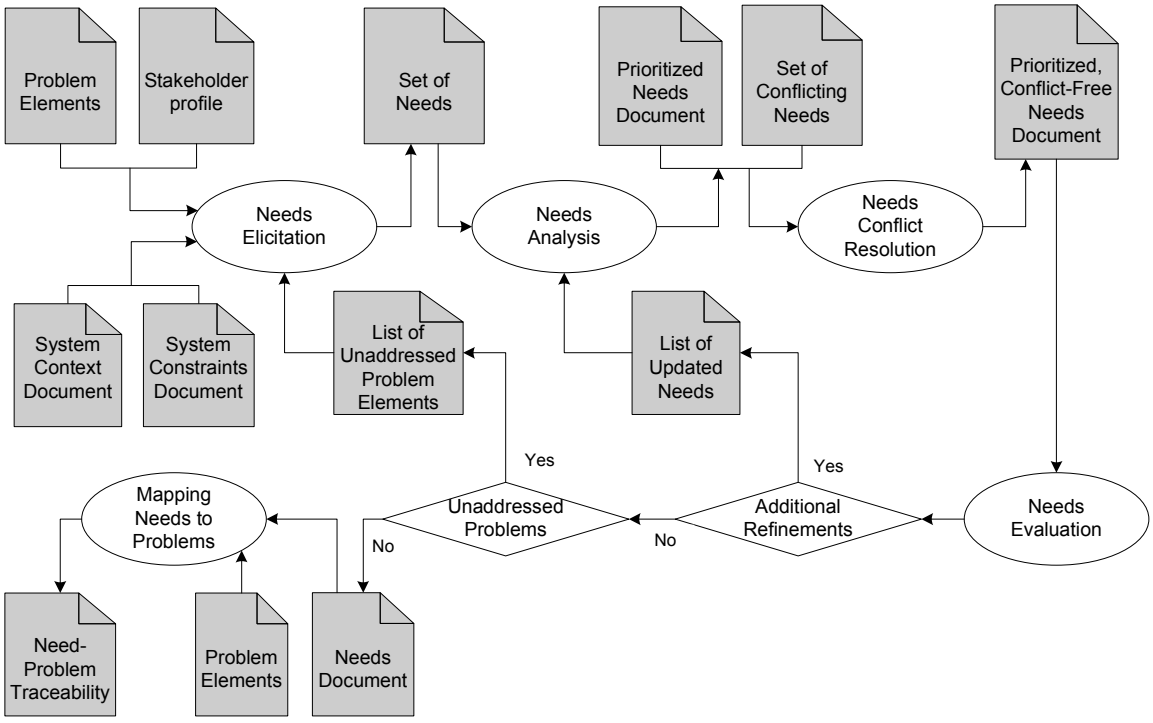



*Figure 3.9 – Needs Generation*

#### 3.2.3.1. Needs Elicitation

Needs Elicitation is the first activity in the needs generation phase. The objective of this activity is to interact with the relevant stakeholders and elicit needs related to those problem elements identified in the Problem Analysis phase. It is important to invest in resources for determining needs as the users often have vague ideas about the problems and needs.

The requirements engineer should consider that the needs are characterized in the problem domain and not the solution domain. In addition, the needs should be captured for each and every problem element. It is also necessary to record the rationale for each elicited need in order to facilitate the tracing of needs to problem elements – this helps in better change management.

Several techniques such as interviews, focus groups, brainstorming, and questionnaires can be used in eliciting user needs. Scenarios are also helpful in providing a clear picture of the customers needs.

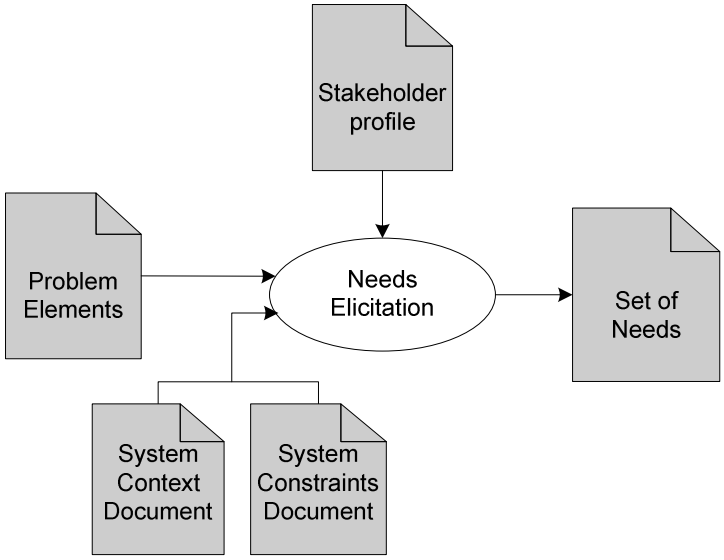



*Figure 3.10 – Needs Elicitation*

As illustrated in Figure 3.10, the input to this activity consists of Problem Elements, Stakeholder Profile, and System Context and Constraint documents. The Problem Elements document drives the Needs Elicitation activity while the remaining documents provide supporting information. The output of this activity is a set of needs, which specifies features that the stakeholders would like to see in the proposed system.

#### 3.2.3.2. Needs Analysis

The objective of the needs analysis phase is to organize and integrate the needs elicited from different sources. In addition, the user needs should be prioritized in order to facilitate a systematic approach to the elicitation of the requirements.

Needs Analysis begins by integrating and documenting the needs elicited from different stakeholders (Figure 3.11). In addition, the rationale of the needs should also be included in the document to facilitate the mapping of the needs to the problem elements. The needs can be grouped based on the system features, user interface, and several other parameters. Organization of the needs can be performed through techniques such as affinity diagrams or functional hierarchy decomposition. The needs are then prioritized to determine the features that are implemented first. The prioritization is performed with the active participation of the customer.

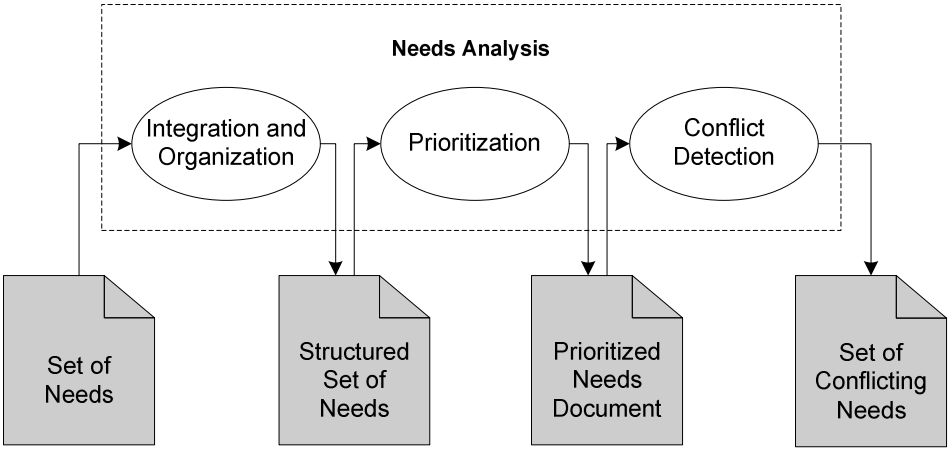



*Figure 3.11 – Needs Analysis*

During Needs Analysis, the requirements engineer also identifies and records conflicts among needs. This document is then passed to the Needs Conflict Resolution activity in which negotiation is used to resolve conflict – stakeholder participation is an absolute necessity.

#### 3.2.3.3. Needs Conflict Resolution

User needs are elicited from different stakeholders; therefore, it is possible to have conflicts among the stakeholder needs. This activity focuses on resolving those conflicts so that an agreeable compromise is reached.

Two approaches are generally used for conflict resolution – position-based and interest-based negotiation. In position-based negotiation, both the parties retain their positions and bargain, whereas in interest-based negotiation, the parties attempt to understand the position of each side and try to reach a solution which is satisfactory to all. Hence, interest-based negotiation is the most widely used in the industry.

Before the negotiation begins, it is necessary for the requirements engineer to prepare for the activity. This involves developing an understanding of the concerns of the customer, and also enumerating the possible solutions to help resolve deadlock situations. The requirements engineer's communication and persuasive skills are important in conducting this activity.

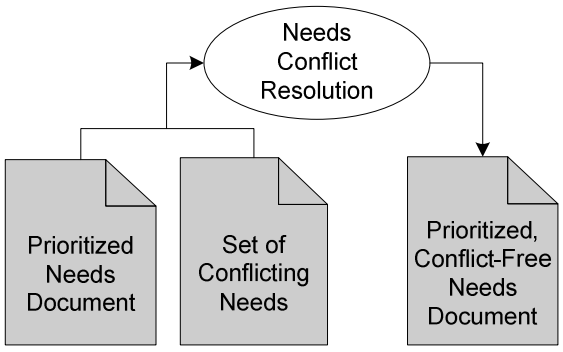



*Figure 3.12 – Needs Conflict Resolution*

The output of this activity is a prioritized and conflict-free list of needs (Figure 3.12). This set of needs is then analyzed during the Needs Evaluation activity to detect possible errors.

#### 3.2.3.4. Needs Evaluation

Needs should clearly state and address all problem elements identified in the Problem Analysis phase. Furthermore, those needs should be internally consistent and conflict-free.

In general, this activity examines the needs for consistency and completeness. In addition, the needs are validated by the stakeholders, that is, determining if the needs address the real problems. Commonly used techniques are walkthroughs and storyboarding. Low-fidelity prototypes, such as paper prototypes, can also be useful in this activity.

The output of this activity is a needs document and a list of needs modifications. On completion of this activity, the requirements engineer decides whether another iteration of this phase is necessary (Figure 3.13). Errors or inconsistencies found during the Needs Evaluation activity are corrected, and an updated needs list is passed to the Needs Analysis activity. In addition, if there are problem elements remaining to be discussed, the Needs Elicitation activity is iterated to derive needs for those problem elements. If no other iterations are needed, the Needs document is passed to the next activity which maps and records needs to the problem elements.

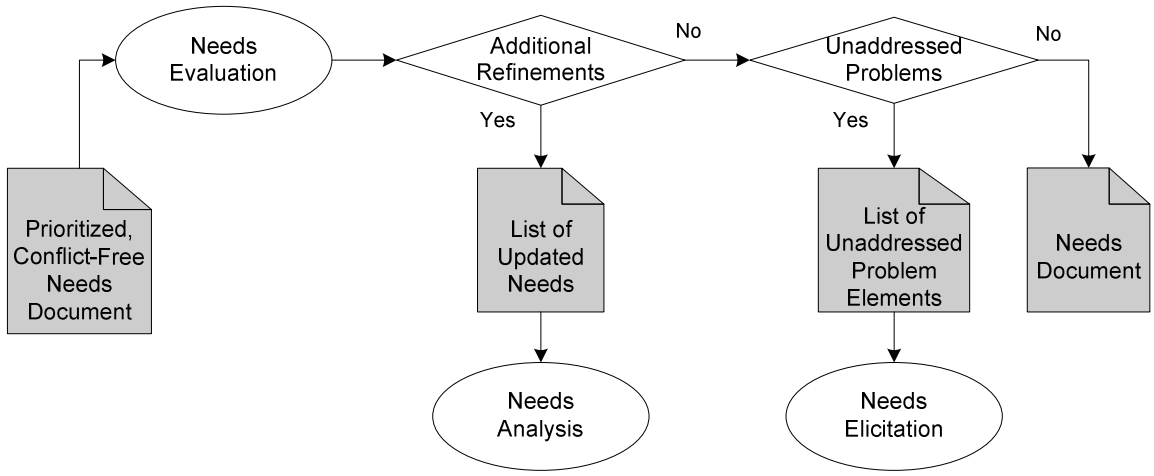



*Figure 3.13 – Needs Evaluation*

#### 3.2.3.5. Mapping Needs to Problems

The objective of this activity is to link the identified needs to the problem elements. In effect, the traceability determines whether all problem elements have been completely addressed by the needs and whether all elicited needs are necessary.

The requirements engineer records the traceability of the need to the problem element based on information obtained from the needs analysis activity. The traceability matrix is a common method for representing the traceability. Furthermore, the traceability can be supported by requirements management tools (Section 2.4.2).

The output of this phase is the Need-Problem Traceability document. This document is useful in managing changes to requirements because it provides the rationale behind the request and an indication of the impact of changes.

After the completion of this phase, the Needs document is passed to the Requirements Capturing phase which elicits requirements based on the needs.

## 3.3. Requirements Capturing

Requirements Capturing involves the elicitation, analysis, verification, and validation of subsets of requirements. This phase begins with the education of the customer on

different aspects of the requirements engineering process. After Customer Education [Arthur 1999], requirements are elicited from the stakeholders based on user needs. Elicited requirements are then analyzed and prioritized according to importance, and evaluated to ensure quality and correctness (Figure 3.14). This phase is iterated until a complete set of requirements is derived.

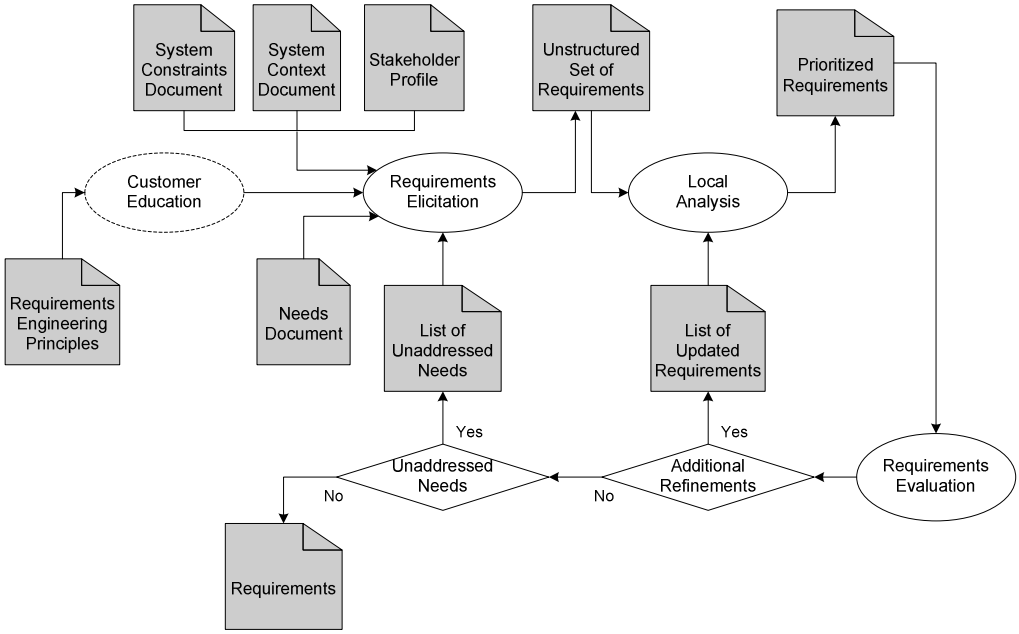



*Figure 3.14 – Requirements Capturing*

The activities in the Requirements Capturing phase are described in the following sections.

### *3.3.1. Customer education*

Customer education is an optional activity. The objective of this activity is to familiarize the stakeholders with the requirements concepts and to emphasize the importance of requirements in the development of a software product. Furthermore, it specifies the role of the customers/users in the requirements engineering process. Thus, this activity improves the understanding of the stakeholders relative to the requirements engineering process and their responsibilities.

The information provided to the customer should include an explanation of the SDLC and the role of requirements engineering in the software development. Furthermore, the requirements engineer should provide an overview of the requirements generation process, and list the responsibilities of the customers and users during the requirements engineering process. In addition, the characteristics of a good requirement are also outlined.

The customers may have limited time and hence, we characterize this activity as a guideline rather than a mandatory activity. Nevertheless, information about the requirements engineering process can be provided to the customer as part of the requirements elicitation meeting.

### *3.3.2. Requirements Elicitation*

The objective of the Requirements Elicitation activity is to correctly elicit and record requirements from the stakeholders. The Needs document is the input to this activity, and drives the elicitation of requirements. Needs differ from requirements in that they are characterized in problem domain terms; requirements, on the other hand, are specified in solution domain terms. Needs are determined from the problem elements; requirements are derived from the needs. Furthermore, there is no formal representation of the needs while several standards are available for representing requirements. The Stakeholder Profile, System Context and Constraints documents are also inputs to this activity (Figure 3.15). These documents are useful in characterizing both the non-functional and functional requirements.

Requirements Elicitation can be performed using several different approaches, e.g., JAD, PD, and so forth. Based on the project characteristics and constraints, the requirements engineer decides which approach to take. Once the approach is selected, the requirements engineer selects the specific techniques for eliciting the requirements. Commonly used techniques include interviews, brainstorming, and focus groups. The output of the Requirements Elicitation activity is an unstructured set of requirements.

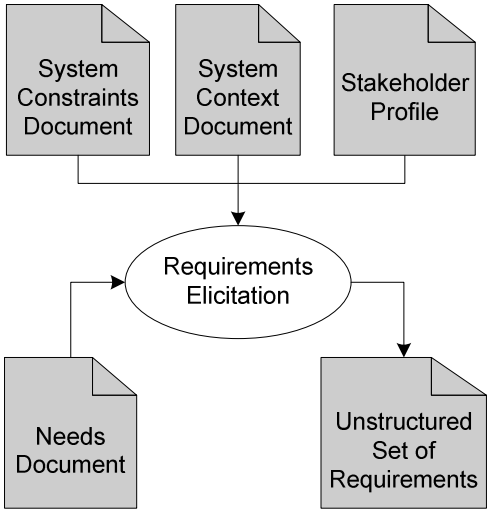



*Figure 3.15 – Requirements Elicitation*

### *3.3.3. Local Analysis*

The objective of Local Analysis is to understand the nature of elicited requirements, and then to structure and prioritize them. This involves modeling requirements to support analysis and comprehension, organizing the requirements, and identifying their particular attributes. Additionally, the requirements are ranked according functionality priorities (Figure 3.16).

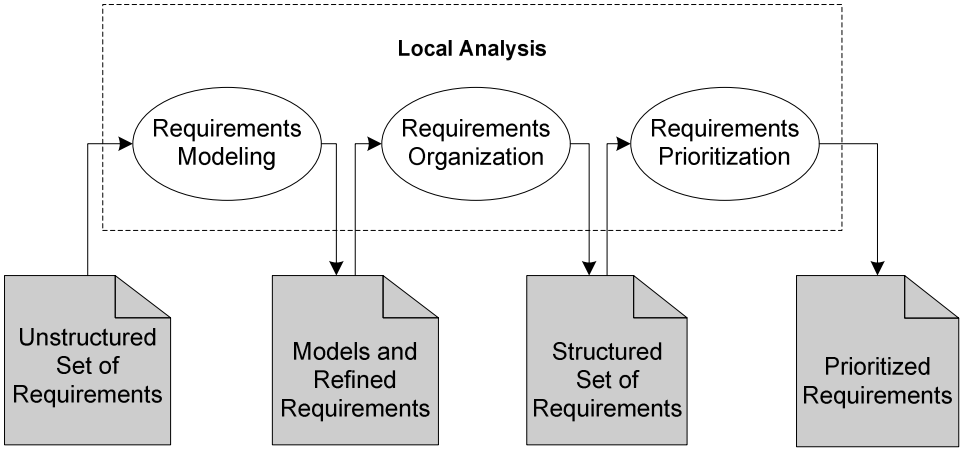



*Figure 3.16 – Local Analysis*

Modeling is needed to develop an understanding of the requirements and represent them in a clear and comprehendible manner [Greenspan 1982]. Various graphical representations, such as data flow diagrams (DFD) and entity-relationship diagram (ERD), are used for modeling[13]. These documents assist in the understanding and formulation of requirements, and are included in the SRS as a supplementary material.

The organization of requirements entails classifying them as functional and non-functional requirements. The functional requirements are further categorized based on the functionalities/features of the system. The non-functional requirements are classified into categories based on performance, security, usability, and so on[14]. The most common techniques used for such classification are affinity analysis and hierarchical decomposition. The resulting (structured) set of requirements should reflect defined formatting standards to reduce the effort in creating the SRS.

Prioritization of the requirements involves identifying the requirement attributes and ranking the requirements according to priority criteria. Several requirement attributes are identified during this activity; these help in the analysis of the combined sets of requirements during the Global Analysis phase. The major requirement attributes are:

- **Risk factors** – specifies the potential risks that may affect the implementation of the requirements, for example, lack of personnel and expertise, budget, and so forth.
- **Effort** – provides a rough estimate of the amount of effort needed in implementing the requirements.
- **Rationale** – identifies the user need from which the requirement is derived, and provides a justification for the requirement.
- **User importance** – specifies the priority of the requirement.

Values for the attributes are determined in consultation with the customer. The requirements are then ranked based on the user importance factor. The prioritized requirements document ranks the structured list of requirements according their

[13] The formats of these representations are presented in Chapter 4.
[14] The detailed discussion of non-functional requirements categories is provided in Chapter 4.

respective high-level functionality. These high-priority requirements are given preference when determining the features that are to be implemented in the software.

### *3.3.4. Requirements Evaluation*

The Requirements Evaluation activity performs the verification and validation on an elicited subset of requirements. Verification and validation is performed at this stage to identify errors earlier in the requirements generation process.

Verification enables the identification of errors relative to particular quality characteristics. Because requirements are elicited in increments, the quality attributes such as completeness cannot be verified. The most commonly verified quality attributes are:

- Ambiguity – requirements should have only one possible interpretation.
- Testability/verifiability – requirements should be testable or verifiable through demonstration or inspection.
- Correctness – requirements should represent some required aspect of the system
- Understandability – requirements should be easily understood by all classes of readers.

Verification can be achieved through techniques such as inspections, audits, and reviews. On completion of this activity, the requirements engineer lists the requirements which do not meet the quality attributes and recommends corrections.

Validation determines whether the requirements capture the customer's intent. If incorrect requirements have been captured, validation identifies these discrepancies. Techniques such as prototyping and storyboarding are widely used for validation. Customer feedback and recommendations are documented.

At the end of the Requirements Evaluation activity, the requirements engineer makes several decisions based on the output of the verification and validation process (Figure 3.17). If the requirements have been modified or refined, then iteration through the Local Analysis activity is performed again. If it is determined that some needs have not been addressed, the Requirements Elicitation activity is revisited. If neither of these situations

occurs, we conclude that all the needs have been addressed and the requirements satisfy quality standards and customer's needs. The Requirements Capturing phase is then exited.

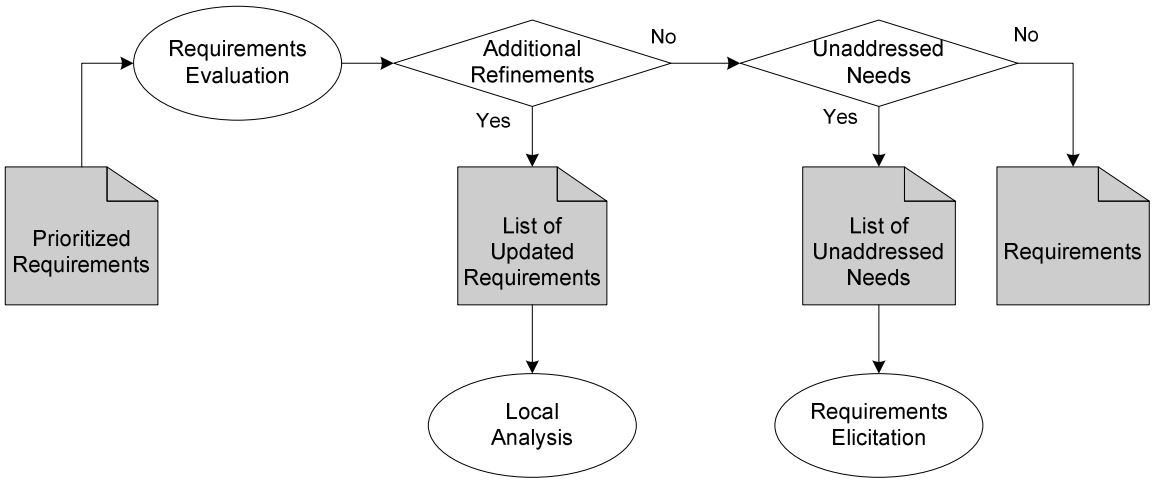



*Figure 3.17 – Requirements Evaluation*

## 3.4. Global Analysis

Upon completion of the Requirements Capturing phase, a complete set of requirements have been generated. That set is then analyzed from several different perspectives. During the Global Analysis phase, the requirements are analyzed for risk, cost and schedule, price, and feasibility. Inconsistencies among requirements are also identified during these analysis activities; conflicts, if found, are resolved through negotiation with and among the stakeholders (Figure 3.18).

The input to this phase consists of the following documents:

- Software Requirements –requirements elicited during the Requirements Capturing phase.
- System Context and Constraints document – specifies the boundaries and restrictions of the proposed system.
- Organizational Standards, Rules, and Protocols – describes the organizational constraints on the system.

- External Influences: These describe the constraints imposed on the system by external systems or environments.

The last two documents are supplementary to the information provided by the System Context and Constraints document.

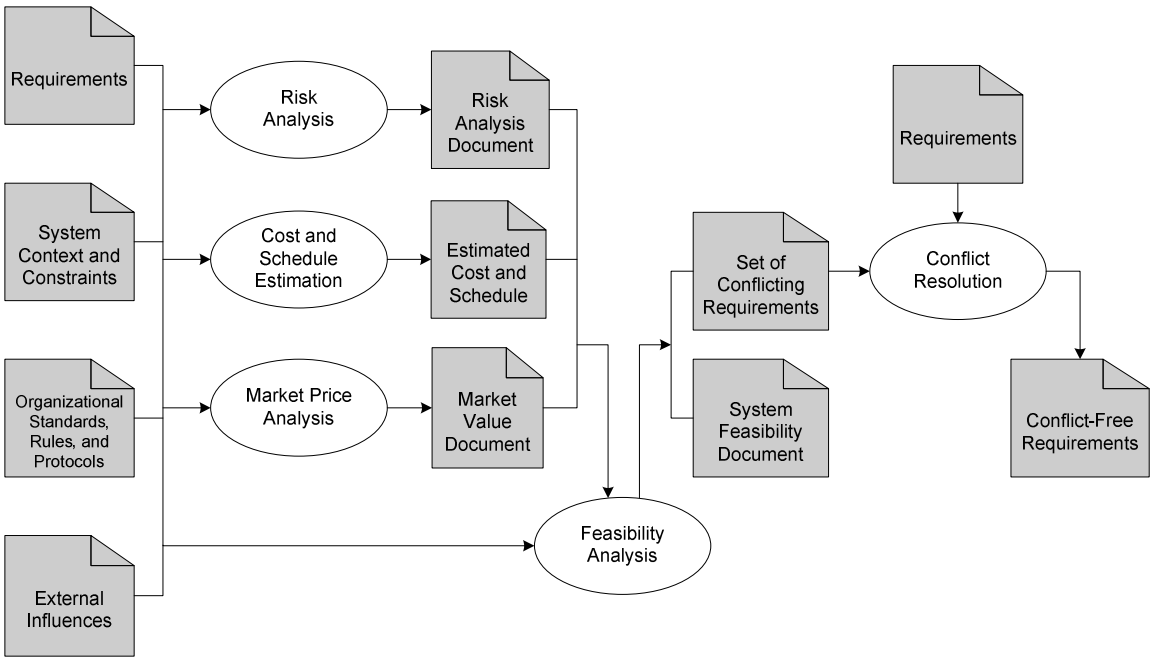



*Figure 3.18 – Global Analysis*

The output of this phase is a feasible and conflict-free list of requirements. In the subsequent sections, we elaborate on the activities involved in the Global Analysis phase.

### *3.4.1. Risk Analysis*

The objective of this activity is to analyze the requirements for potential risks, and to determine the high risk requirements. Risks are commonly categorized as [SEI-Risk 1996]:

- **Product engineering** – risk factors related to technical aspects of the product.
- **Development environment** – risk factors related to work environment, development processes, and management practices.

- **Program constraints** – risk factors related to stakeholders, resources, and contracts.

During Risk Analysis, the requirements engineer determines the probability of the risk factor occurring, and the possible loss if that risk factor materializes. The product of the values provides the estimate of risk exposure. Any requirement having a risk estimate more than a critical value, predetermined by the management, is classified as a high-risk requirement.

The output of this activity is a Risk Analysis document which records the risk estimate of individual requirements. In addition, the document may also contain recommendations for mitigating high risks.

### *3.4.2. Cost and Schedule Estimation*

This activity determines the cost and time required to implement the requirements. These estimates are critical as they affect the budgeting, planning, and other management decisions.

Cost estimation involves determining the expected cost to be incurred in developing individual components of the software. During this activity, the profit component of development is ignored; the sole purpose is to determine the effort involved. Several techniques such as COCOMO, function point analysis, and work breakdown structure are commonly used.

Schedule estimation identifies the time needed to implement the different components of the product. The schedule estimate helps management in the distribution of resources so that the project can be completed on time. Several techniques such as PERT charts and the CPM (critical path method) are used estimate project schedule. Schedule estimates are also determined in consultation with the developers.

The final output of this activity is a Cost and Schedule Estimate document which lists the cost and time needed for developing the different components/requirements of the product.

### *3.4.3. Market Price Analysis*

Market Price Analysis is used to determine if a proposed price for the product is reasonable and fair with respect to the market demand. It estimates an appropriate price for the product by balancing the functionalities of the system with the actual user needs.

Before determining whether the price is reasonable, it is necessary to conduct a market survey (Figure 3.19), which collects information about the demand for the product, competing products and their functionalities, the most desirable features, price of the similar products, and so on. This information helps in proposing a viable price.

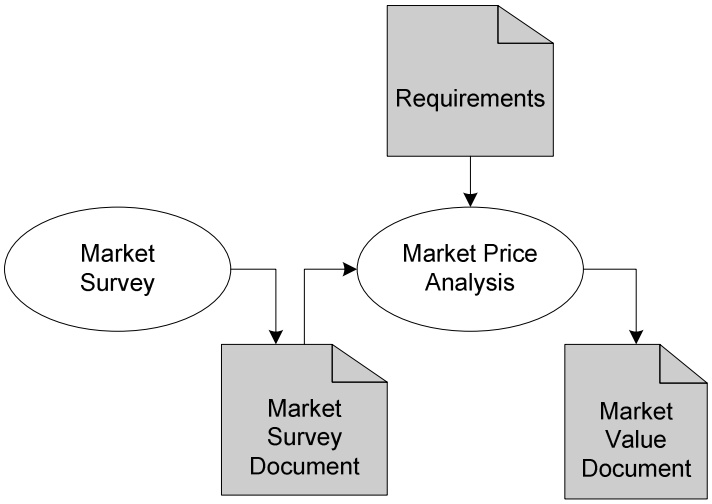



*Figure 3.19 – Market Price Analysis*

The output of this activity is a Market Value document which states the appropriate price for the product.

### *3.4.4. Feasibility Analysis*

The objective of the Feasibility Analysis activity is to determine how beneficial or practical the development of the project is to the organization. In order to perform this analysis, the outputs generated from the previous Global Analysis activities are used as inputs. The output of this activity is critical for management decisions.

Feasibility analysis involves checking the requirements from five perspectives:

- **Operational feasibility** – determines how well the requirements satisfy the business objective, and whether the users can easily adapt to the system.
- **Technical feasibility** – determines the capability of the technical team in developing the product.
- **Schedule feasibility** – the schedule estimate is examined for practicality.
- **Economic feasibility** – determines if the development of the product produces sufficient profits in the long term.
- **Legal feasibility** – is the process of determining the legal consequences of implementing the requirements.

During feasibility analysis the requirements engineer identifies inconsistent or infeasible requirements. These requirements are added to the list of conflicting requirements compiled in the previous activities – risk, price, cost, and schedule analysis. At the end of this activity, the requirements engineer prepares the System Feasibility document. Additionally, the requirements engineer also prepares a list of conflicting requirements which must be resolved during the next activity, Conflict Resolution.

### *3.4.5. Requirements Conflict Resolution*

This activity involves resolving the identified requirement conflicts. The requirements engineer and stakeholders meet and employ negotiation strategies to resolve the misunderstandings and conflicts.

Conflict Resolution activity consists of identifying why the requirements conflict, determining alternative solutions, and negotiating an agreement. This activity does not always result in an agreement. If a deadlock situation occurs, the requirements engineer has to contact the customer's higher level management. If this does not result in an agreement, the requirements engineer consults higher-level management authorities until some resolution is achieved. Again, position-based or interest-based negotiation can be used for conflict resolution.

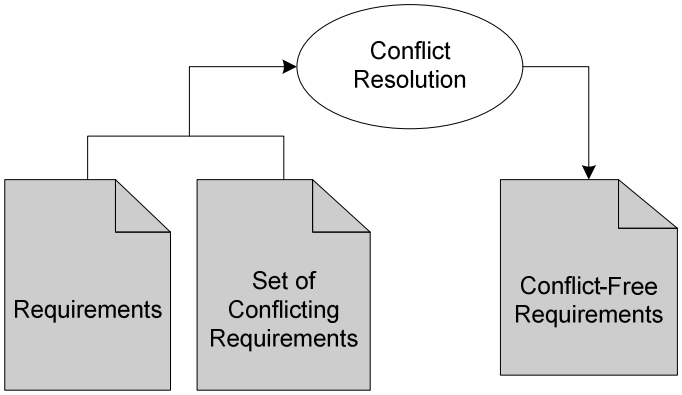



*Figure 3.20 – Requirements Conflict Resolution*

The output of this activity is a Conflict-Free Requirements document (Figure 3.20) and is passed to the Requirements Specification phase for producing the SRS.

## 3.5. Requirements Specification

Once the complete set of requirements is obtained and the conflicts are resolved, the requirements are then organized to form the Software Requirements Specification (SRS) (Figure 3.21). The SRS is the basis for communication among clients, end-users, system designers, and developers of the software. Thus, the SRS represents the user requirements, serves as a contract between the development team and the customer, and provides information to the developers for designing and implementing the system. The SRS must be internally consistent, complete, and unambiguous.

In order to clearly describe all the requirements, the specification is written according to a standardized format, and contains supporting information to assist in the understanding of the information. There are variety of formats and styles, including text and graphics, for the presentation of the SRS. The commonly used approach to represent requirements in the SRS is listing them in a textual form. The inclusion of graphical representations facilitates comprehension of the SRS. Templates for the SRS are provided in Appendix B.5.

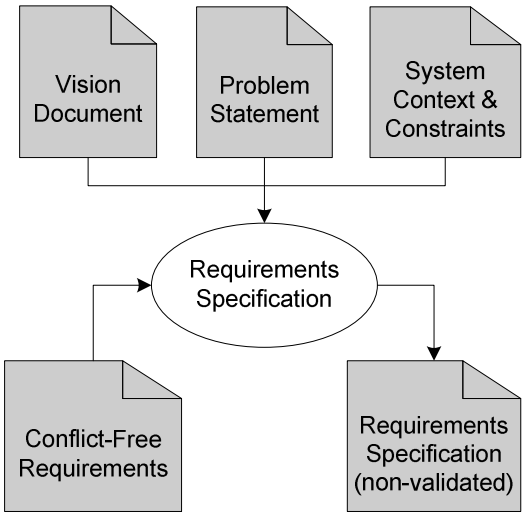



*Figure 3.21 – Requirements Specification*

In addition to functional and non-functional requirement statements, the SRS contains a variety of technical and non-technical text, such as:

- Description of the environment and objectives of the system
- Definition of the scope of the requirements
- Stakeholder descriptions
- Supplementary models used in deriving the requirements, such as DFD and ER diagrams.

In the past, requirements have most often been documented and modified within a software requirements specification. However, in recent years requirements are starting to be captured and maintained in a database rather than a document. Several management tools are available for creating and maintaining such a database (Section 2.4.2).

## 3.6. SRS Evaluation

This last phase focuses on the verification and validation of the SRS. During this phase, the SRS is evaluated for quality characteristics. In addition, the contents of the SRS, in

particular, the requirements, are validated by the customers/users (Figure 3.22). Upon completion of this phase, the SRS is ready to be signed by the customer and baselined.

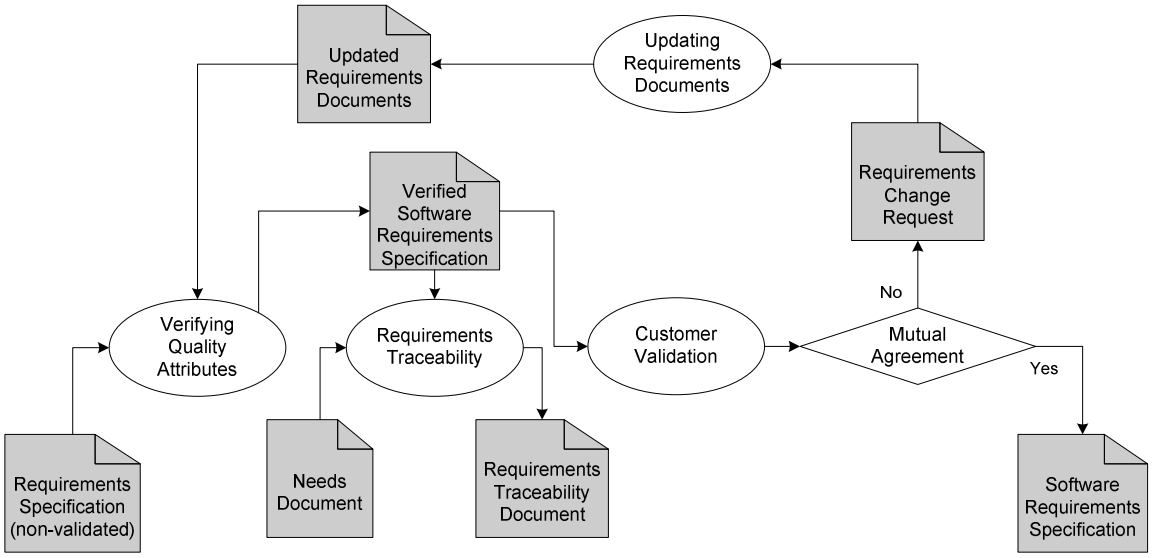



*Figure 3.22 – SRS Evaluation*

The activities in this phase are described in the following sections.

### *3.6.1. Verifying Quality Attributes*

The objective of this activity is to check whether the SRS satisfies the prescribed quality standards. This activity is different from the verification activity in the Requirements Capturing phase as it evaluates the SRS, which consists of the project purpose, complete set of requirements, and so forth (Section 4.1.5.1), rather than a subset of the requirements. This activity is often conducted by a quality expert.

The requirements engineer (quality expert) verifies the SRS for several quality characteristics, some of which are described below:

- Consistency – SRS is consistent if there is no conflicting information.
- Completeness – "SRS is complete if everything that the software is supposed to do is included" [Davis 1993].
- Unambiguous – Contents in the SRS, especially requirements, should be clear and have only one possible interpretation.

- Modifiability – The SRS should be written so that further changes are easily incorporated into the document.

The output of this activity is the verified SRS, which includes clear and precise information and conforms to standards prescribed for the SRS.

### *3.6.2. Requirements Traceability*

After the SRS, including the complete set of requirements, is checked for quality attributes, a formal requirements traceability document is created and verified. This activity generates the Requirements Traceability document by using linkage information captured in the earlier phases. The traceability document facilitates representing relationships of requirements and the rationale for generating the requirements. In addition, this document is useful for requirements change management process.

Requirements are often dependent on the existence of other requirements and these relationships need to be documented. Furthermore, the requirements should also be traced back to the user needs. This type of traceability identifies the rationale of the requirements and determines if the requirement is necessary.

Traceability matrices and trees are the common techniques for representing the traceability of requirements. Furthermore, traceability is often facilitated by the use of requirements management tools such as DOORS, RequisitePro, CaliberRTM, and so on.

### *3.6.3. Customer Validation*

The objective of this activity is to ensure that the final set of requirements documented in the SRS reflect the actual user needs. This activity is different from the validation activity conducted in the Requirements Capturing phase as it evaluates the complete SRS rather than the subsets of requirements.

This activity can be performed using any one or a combination of several techniques, e.g., storyboarding, prototyping, walkthroughs, and so forth. The stakeholder feedback is documented so that the necessary changes can be made to the final set of requirements.

After stakeholder validation, the requirements engineer has to decide whether the final set of requirements in the SRS meet the customer's intent (Figure 3.23). If changes are needed based on the customer feedback, the SRS is updated and evaluated again. Otherwise, it indicates that the SRS is ready for baselining.

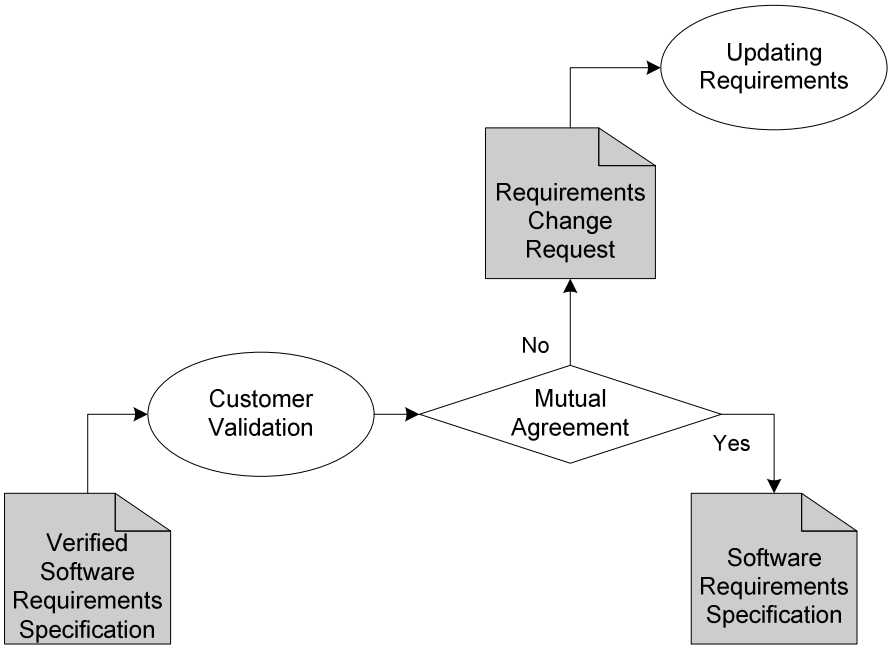



*Figure 3.23 – Customer Validation*

### *3.6.4. Updating Requirements Documents*

As a result of the validation, if the requirements need to be updated, the required changes are documented. In this activity, the potential changes are analyzed, negotiated, and incorporated into appropriate requirements documents, such as the SRS and requirements traceability.

Once the requirements documents are updated, it is necessary to iterate through the Requirements Verification and Validation phase once again to ensure that the updated SRS follows quality standards and meets the customer's intent.

**Summary:** In this chapter, we focus on the decomposition of the RGM so that it reflects an adequate level of abstraction to represent the flow and evolution of requirements synthesis. The model begins with the Conceptual Overview phase, which captures the

customer's perspective of the proposed system. This is followed by the Problem Synthesis phase, where the actual problem and root cause is identified or confirmed. The high-level problems are then decomposed into problem elements, and the user needs associated with each problem element are identified. The Requirements Capturing phase elicits, analyzes, and evaluates sets of requirements corresponding to the needs. After capturing the complete set of requirements, we then evaluate the requirements from a global perspective. The requirements are then documented in the SRS in the Requirements Specification phase. The Verification and Validation phase then ensures the quality and correctness of the SRS. Once the SRS is approved by the customer/user, it is baselined according to the organization's configuration control policy. The baselining and version control of the SRS is not discussed in this model, as the focus of this research is on the generation of the SRS, and not the change management aspect of the requirements.

In the next chapter, we discuss the characteristics of the intermediate and final requirements documents. We also present the transformation of those documents as requirements evolve through the requirements engineering process.

# CHAPTER 4 – EVOLUTION OF THE REQUIREMENTS DOCUMENTS

## 4. Introduction

Chapter 3 describes the refinement of the requirements generation process into a sequence of activities that facilitates the representation of the requirements flow. This chapter presents the detailed characteristics of the requirements documents identified in Chapter 3, and illustrates the transformation of requirements starting from the elicitation of user ideas and extending to the generation of the SRS.

Chapter 4 is organized into two parts – (1) a discussion of the detailed characteristics of the requirements documents, and (2) a discussion of the transformation of these documents during the requirements generation process. For each document, Section 4.1 describes the document's objective, content, possible formats, and its synchronization with related activities. The recommended formats for the requirements documents are provided in Appendix B. Section 4.2 illustrates the transformation of requirements documents using graphical representations, and describes how information is passed through those documents as one progresses through the requirements generation process.

## 4.1. Requirements Documents

Documents form an essential part of the requirements engineering process because they represent information that is elicited or generated. In addition, documents are used as communication tools among stakeholders. In the sections that follow, the characteristics of documents produced in the different phases of the requirements generation process are discussed in detail. The emphasis is on identifying the content and format of the requirements documents to support the related activity objectives.

### *4.1.1. Conceptual Overview Documents*

The requirements generation process begins with the Conceptual Overview phase (Appendix A.1) which is focused on capturing the customer's viewpoint concerning the

problem and the proposed system. Conceptual Overview also identifies the need for the new system from a business and operational perspective, and ensures that all stakeholders have a common understanding of project objectives. In order to generate a common product vision, the inadequacies of the current system are analyzed, and the benefits of the new system are justified through customer meetings. The product objectives and the vision together form the basis for the requirements.

The Conceptual Overview phase generates two documents – Customer Perspective of the Problem and the Vision document. During the initial meetings, the customer's portrayal of the problems and desired solution is recorded in the Customer Perspective of the Problem. Based on this document, the Vision document is created, which highlights problems and justifies the development of the proposed system.

The input to the Conceptual Overview phase can be HLR, ConOps, or user ideas. These documents may not be present for all projects. However, if these documents do exist, they simplify the initial activities of the requirements generation process.

#### 4.1.1.1. HLR (High Level Requirements)

When the software system is a part of a larger system, the system engineering process conducted prior to software development, generates the HLR. System engineering determines software and hardware subsystems and identifies major capabilities of those subsystems. Therefore, the HLR includes requirements of the high-level functionalities of the software and its interfaces to other parts of the system.

The HLR document provides the overview of the proposed system and defines the purpose and the scope. This information is useful in generating the Vision document and in defining the system boundary [IEEE 1995]. Furthermore, the set of problems inherent to the current system (included in the HLR) forms the basis for the Problem Statement document. Additionally, high-level system capabilities, description of different system states, and major system constraints are included.

The format of the HLR document is a structured document in a natural language that explains the system overview and functionalities (Appendix B.1.1). Diagrams are

included to illustrate the system context, major functionalities of the system, and the interaction of the system with the environment.

#### 4.1.1.2. ConOps (Concept of Operations)

The ConOps document is a document where the users describe their expectations of the target system in application domain terms. This document is produced based on user analysis of the current system and describes system characteristics for the proposed system from the users' viewpoint. "The primary goal for a ConOps document is to capture user needs and express those needs in the user's terminology" [IEEE 1998].

The ConOps document provides a brief description of the current system. The document also includes the justifications for and nature of changes to the current system. Justification can be achieved through high-level goals like increased profits, ease of use, minimized error levels, and so on. Furthermore, the user ideas about the functionalities of the proposed system are listed (Appendix B.1.2). User ideas are supported through scenarios of the system operation. The ConOps document also analyzes the impact of the proposed system and describes the benefits of implementing and deploying the proposed system [Fairley 1996].

The ConOps document is described in the user vernacular. Graphical representations should be used wherever possible to facilitate understanding by different types of readers. Useful graphical tools include, but are not limited to, work breakdown structures (WBS), functional flow block diagrams, structure charts, allocation charts, dataflow diagrams (DFD), context diagrams, storyboards, and entity-relationship diagrams [IEEE 1998].

#### 4.1.1.3. Ideas

It can also be the case that the problems and desired solutions are not recorded in a document. In such situations, the customers/users usually have their own perception of both the problem and the solution, and they articulate their ideas in application domain terms. It is the responsibility of the requirements engineer to capture the customer's ideas and document them. This is usually achieved through the Initial Customer Meeting activity.

#### 4.1.1.4. Customer Perspective of the Problem

During the Initial Customer Meeting (Appendix A.1), the requirements engineer captures the customer's viewpoint on the problem, desired solution, and rationale. The observations made by the requirements engineer are recorded in the Customer Perspective of the Problem document. This document assists in identifying the success factors of the project and provides an initial idea of the rationale and objectives of the project.

The Customer Perspective of the Problem document should include customer's perception of the current system/situation and its limitations. In addition, the requirements engineer should record the user's ideas about the solution. Furthermore, the rationale for developing the proposed system should be elicited from the user and recorded. The rationale helps in determining whether the real needs of the users are being addressed by the proposed system.

This document is written in a natural language and consists of a question-answer format. The proceedings of the meeting can be documented during the activity or it can be completed later by referring to the video/audio recording of that meeting.

The elicitation and documentation of user perspectives is assisted by pre-determined questions, such as:

- What are the problems faced in the system?
- What are the possible solutions to overcome them?
- What are the short term benefits of this?
- How do you vision the benefits of the new system in the long term?
- What are the losses of the proposed system?

#### 4.1.1.5. Vision Document

The Vision document provides a common understanding of the proposed system among the stakeholders. This document is created based on the customer perspective

information. The Vision document acts as a reference for the subsequent activities as it provides an overview of the problem and rationale of the proposed system development.

The Vision document begins with the vision statement which summarizes the long term purpose and intent of the proposed system. The vision statement is usually written similar to the following template [Moore 1991]:

- For [Target customer]
- Who [statement of the need]
- The [product name]
- Is [a product category]
- That [key benefit, compelling reason to buy]
- Unlike [primary competitive alternative, current system]
- Our product [advantages of proposed product/system].

The Vision document provides a brief description of the current system and explains the limitations of the system. The document also provides an overview of the proposed system and its basic functionalities. In addition, the Vision document describes the justifications for the proposed changes by listing the benefits of the new system. Furthermore, the impact of the system on the organization, in terms of change in user expertise, policies, and so forth, are described.

The Vision document describes the overview of proposed system and its rationale in the user's vernacular and in a textual format illustrated in Appendix B.1.4.

### *4.1.2. Problem Synthesis Documents*

On completion of the Conceptual Overview phase, the high-level description of the problem, possible solution, and its rationale is identified. The next phase is Problem Synthesis (Appendix A.2), which performs comprehensive analysis of the current problems and the customer needs. This phase involves identifying the real problems in the current practice and the root cause behind those problems. In addition, this phase derives the needs for identified problem elements.

The Problem Synthesis process starts by identifying the high-level problems (Problem Statement), and then attempting to identify the root cause(s). Based on the root causes, complex problems are decomposed into a set of smaller, distinct problem elements (Problem Elements). In addition to analyzing the problems, the requirements engineer also determines the system boundary and constraints (System Context and Constraint Document). Once the problem elements are identified, user needs are generated by analyzing the problem elements while recognizing system context and constraints. Needs are then prioritized and evaluated by the stakeholders to produce the Needs document.

#### 4.1.2.1. Domain Model

Before eliciting the user needs, the requirements engineer must be educated about the application domain concepts and the relevant artifacts. This objective is achieved through the Requirements Engineer Education activity (Appendix A.2.1). Based on the information obtained during this activity, the requirements engineer creates a Domain Model document describing the application domain, which can be used for the Problem Analysis activities.

Modeling key concepts or business entities and their relationships helps refine the requirements engineer's understanding of the system. It also helps the requirements engineer uncover subtle complexities of the system, as well as to establish a common vocabulary. In domain modeling, the application domain, its components and environment are graphically represented and elaborated (Appendix B.2.1). The model attempts to capture the domain information and is used in the later phases of the requirements generation process.

#### 4.1.2.2. Problem Statement

The Problem Statement is generated by the Problem Identification activity (Appendix A.2.2) which takes the Domain Model and Vision documents as inputs. This activity involves investigating and identifying the real problem in order to support a thorough analysis of the problem. Based on observation of work places and interviews with the stakeholders, the Problem Statement is created describing the problems in the current system, affected stakeholders, and the benefits of resolving the problem.

This document serves as an agreement on the definition of the problem to be solved [Leffingwell 2000]. It is usually composed of those elements listed in the table shown below:

| **Element** | **Description** |
|---|---|
| The Problem of | *Describe the problem* |
| Affects | *List of stakeholders affected by the problem* |
| The result of which | *Impact of this problem business activities* |
| Benefits of resolving | *Indicate the proposed solution and list a few key benefits* |

*Table 4.1 – Problem Statement Description*

The real problem (the root cause of the problem symptom) can be identified by using root cause analysis and Pareto charts. The elicited problems and their causes are often represented by a fishbone diagram (Figure 4.1).

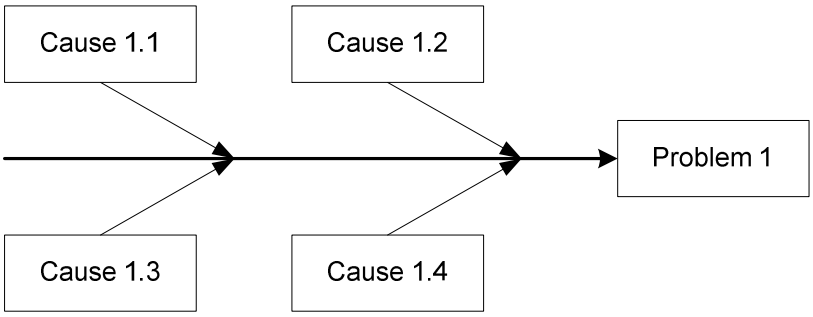



*Figure 4.1 – Fishbone Diagram of Root Causes*

The causes of the problem represented in the fishbone diagram can be further analyzed by using a Pareto chart[15] to determine the factors which cause the problem(s). Pareto charts are used to graphically summarize and display the importance of the problem causes relative to one another (Figure 4.2).

[15] Refer to http://erc.msh.org/quality/pstools/pspareto.cfm

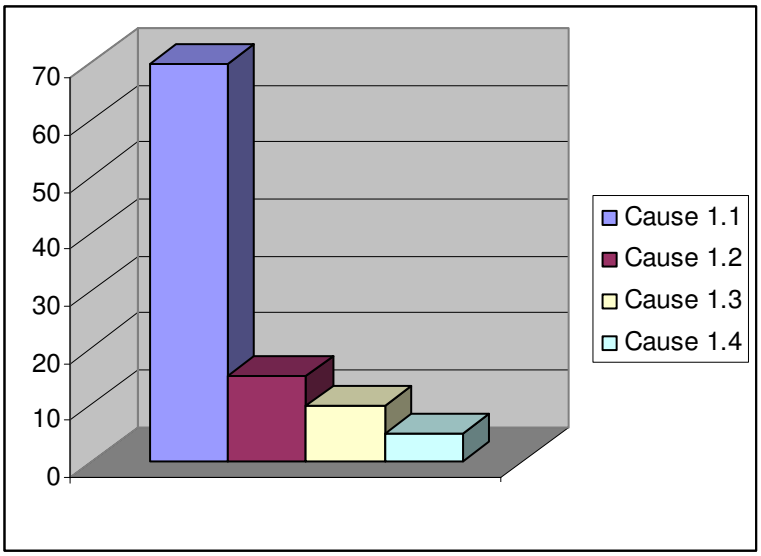



*Figure 4.2 – Pareto Chart of Root Causes*

The problem statement document is written in a textual format and is often supported by illustrations such as Pareto charts, and fishbone diagrams.

#### 4.1.2.3. Stakeholder Profile

In addition to the Problem statement, the Problem Identification activity also produces the Stakeholder Profile document. The Stakeholder Profile document identifies the parties affected by the problems and provides information such as individual responsibilities and application domain knowledge. This document is used as a reference for communication when problems arise in the later activities.

In order to identify stakeholders, the following information must be collected [Leffingwell 2000]:

- Who are the users of the system?
- Who is the customer (economic buyer) for the system?
- Who else will be affected by the outputs the system produces?
- Who will evaluate and approve the system when it is developed and deployed?

- Are there any other internal or external users of the system whose needs must be addressed?
- Who will maintain the new system?
- Is there anyone else?

The content of the document includes stakeholders' names, contact information (postal address, phone number, and email address), their positions and their departments. The position information provides the rank and decision making level of the person. Also, information about the class of knowledge and the expertise level is recorded. For example, class of knowledge can be business goals, business operations, maintenance of the system, and so on.

The format can be either sequential or tabular (Appendix B.2.3). Tabular format is preferred because it is easy to obtain an overview, compare different stakeholders and group them into various categories, e.g., department or class of knowledge.

#### 4.1.2.4. Set of Problem Elements

The Set of Problem Elements document is generated by the Problem Decomposition activity (Appendix A.2.2) whose input documents are the Problem Statement and Stakeholder Profile. This activity involves decomposing the complex problems into its constituent problem elements. Problem Decomposition can be achieved through the use of fishbone diagram (Section 4.1.2.2).

The Set of Problem Elements document consists of smaller, distinct problems. For each problem element, the document provides a brief description of the problem, the affected stakeholders, the impact and the benefits of the solution. Thus, this document is similar to the Problem Statement document, but each problem is at finer levels of detail.

This document uses a textual format for describing the problem elements and is shown in Appendix B.2.4. The textual description can also be supplemented with the graphical fishbone diagram.

#### 4.1.2.5. Problem Elements

Once the Set of Problem Elements is identified, they are structured and examined in the Problem Elements Analysis activity (Appendix A.2.2). The output of this activity is the Problem Elements document.

In this document, the Problem Elements are structured based on their importance, which are assigned by the customer/user. The Problem Elements can be ranked relative to one another, or they can be categorized into priority groups. In addition, the problem elements should be stated clearly to prevent misinterpretations. Hence, this document has similar content and format as the Set of Problem Elements document, but it includes a complete set of ranked problem elements (Appendix B.2.4).

#### 4.1.2.6. External Influences

It is necessary to understand the effects of the environment on the system in order to define the system constraints and scope. For example, an environment influence can be in the form of stipulations to which the system must comply. As another example, it may be necessary to follow a particular standard in order to make the system compatible with other external systems.

The content and format of these documents differ for each organization. However, this document is generally written in a textual format and specifies the categories of the external influences.

#### 4.1.2.7. Organizational Standards, Rules, and Protocols

Every organization has its own internal policies and standards that are adhered to by the various projects. For example, the quality and version control processes differ among organizations. It is the responsibility of the requirements engineer to study these documents to help gain an understanding of the system context and constraints.

As with the External Influences document, the Organizational Standards, Rules, and Protocols documents, often vary from one organization to another. Hence, the

recommended guideline is that these documents should be clear and precise using illustrations wherever it is necessary.

#### 4.1.2.8. System Context Document

The System Context document is produced by the Context and Constraints Analysis activity (Appendix A.2.2) which takes the following input documents – Vision document, Problem Element, Domain Model, External Influences, and Organizational Standards, Rules and Protocols. The System Context document specifies the boundaries within which the proposed system has to operate. This document is important because it helps in identifying the inputs and outputs of the system. Furthermore, it is used to determine the interfaces between the proposed system and its environment.

The System Context document includes information about the external entities/actors interacting with the system. It describes the function and responsibilities of each of these entities. Moreover, this document also describes the data exchanged between the proposed system and the external systems. Context diagrams are often used in this document to graphically illustrate the boundaries and connections between the system and its environment. In addition, it lists the projects priorities, which help in the managerial decisions during the requirements generation process. Five dimensions that are usually considered for this purpose are features, quality, schedule, cost, and staff [Wiegers 1996].

This document consists of context diagrams which are supported by textual descriptions and provides a better understanding of the system and its environment (Appendix B.2.5).

#### 4.1.2.9. System Constraints Document

The Context and Constraints Analysis activity also produces the System Constraints document, which specifies the restrictions on the proposed system. Types of constraints include technical, political, design, legal, and so on. The requirements engineer needs to identify the various types of constraints, analyze them and then identify the potential constraints. These constraints are critical because they contribute to the non-functional requirements in the SRS.

Constraints should be grouped based on the following constraint types [Leffingwell 2000]:

| Constraint Type | Sample Considerations |
|---|---|
| Economic | Are there licensing issues? |
| Political | Interdepartmental problems or issues? |
| Legal | Are there regulatory constraints? |
| Design | Which operating system should be used? |

*Table 4.2 – Categories of System Constraints*

It is also beneficial to identify the rationale for the constraint to ensure that the scope of the constraint is understood. It also helps in identifying when the constraint is not applicable.

The Systems Constraints document is usually written in tabular format as shown in Appendix B.2.6.

#### 4.1.2.10. Set of Needs

The Set of Needs document is generated by the Needs Elicitation activity (Appendix A.2.3), which has the following input documents – Problem Elements, System Context and Constraints, and Stakeholder Profile. The Needs Elicitation activity involves interacting with the relevant stakeholders and eliciting needs for the problem elements. The output (Set of Needs document) of this activity specifies the features that the stakeholders would like to have in the proposed system. It is the responsibility of the requirements engineer to ensure that needs are elicited for all the problem elements.

The Set of Needs document consists of needs which are categorized on the basis of the problem elements. The documented needs should be clear and precise. Combining multiple needs into a single statement should also be avoided. In addition, the needs should use terms consistently to prevent misunderstanding and ambiguity. To facilitate the tracing of needs to problem elements, this document also specifies the rationale for the needs.

The Set of Needs document is recorded in textual format and organized according to the problem elements (Appendix B.2.7). Scenarios can be used to supplement the needs.

#### 4.1.2.11. Structured Set of Needs Document

The Needs Elicitation activity elicits sets of needs from various stakeholders. These needs can be unorganized and often difficult to comprehend. Hence, the Structured Set of Needs document integrates and organizes the elicited needs. The organization of the document is based on the high-level features which are identified during the Needs Analysis activity (Appendix A.2.3). In addition, this document also includes the rationale for the needs.

The format for the Structured Set of Needs document is illustrated in Appendix B.2.8.

#### 4.1.2.12. Prioritized Needs Document

Once the needs are structured, the Needs Analysis activity prioritizes the needs. The input for Needs Prioritization is the Structured Set of Needs document. Prioritization of the needs assists in identifying the critical needs that are to be implemented first.

The Prioritized Needs Document consists of needs which are ranked based on their importance to the stakeholders. Needs can be ranked relative to one another, or they can be categorized into priority groups. The priority level of the needs is often included in the document as an attribute of the needs.

The content and format of this document is similar to the Structured Set of Needs document but includes an additional priority attribute (Appendix B.2.9). In addition, the needs in this document are rearranged according to the priority attribute. Priority levels of the needs can be shown by a numbering scheme or as a characteristic of the needs.

#### 4.1.2.13. Set of Conflicting Needs

The Needs Analysis activity also produces the Set of Conflicting Needs document, which records the conflicting needs identified during Needs Analysis. This document is important because it identifies inconsistencies in the needs.

This document is composed of conflicting needs and the reasons for those conflicts (Appendix B.2.10). In addition, the possible solutions for the conflicts can also be included. The conflicts are described in the vernacular of the user so that the stakeholders can better understand conflicts, and hence, contribute more effectively during the resolution of those conflicts. If additional information about the conflicting needs is required, the requirement engineer often refers to the Prioritized Needs Document.

#### 4.1.2.14. Prioritized, Conflict-Free Needs Document

The Prioritized, Conflict-Free Needs Document is generated by the Needs Conflict Resolution activity (Appendix A.2.3), whose input documents include the Prioritized Needs document and the Set of Conflicting Needs. This activity resolves the conflicting needs and incorporates the agreed-to solutions into the needs document to generate the Prioritized, Conflict-Free Needs Document.

The content and format of the Prioritized, Conflict-Free Needs Document is the same as the Prioritized Needs document, the difference being that the needs are conflict-free (Appendix B.2.11). In the case there are a large number of needs, managing changes to the needs is difficult. In such situations, it is recommended to store the needs in a database, which can be manipulated by management tools.

#### 4.1.2.15. List of Updated Needs

The Needs Evaluation activity (Appendix A.2.3) produces the List of Updated Needs document based on the stakeholder feedback on the Conflict-Free Needs document. This document lists the modifications/refinements of the needs.

The List of Updated Needs document has similar contents as the Structured Set of Needs document. However, some additional information is included. This document describes the modified need and the reasoning behind that modification (Appendix B.2.12). Furthermore, the stakeholder requesting the change is also listed so that s/he can be contacted for clarifications. Several other attributes such as date of modification, impact of the change can also be included.

#### 4.1.2.16. List of Unaddressed Problem Elements

On completion of the Needs Evaluation activity, the requirements engineer checks whether all the problem elements have been addressed by the elicited needs. If s/he determines that some problem elements have not been discussed with the stakeholders, then those problem elements are appropriately marked and passed to the needs elicitation activity.

This document is a subset of the Problem Elements document; therefore, it has similar content and format. The only difference is the inclusion of the list of unaddressed problem elements. This distinction can be achieved through a coloring scheme (text color or highlight) or categorization of needs (addressed/unaddressed needs).

#### 4.1.2.17. Needs Document

The Needs document is the final output of the Problem Synthesis phase and is generated by the Needs Evaluation activity. If the needs reflect the stakeholders' intent and address all the problem elements, it indicates that the complete set of needs have been elicited. This set of needs is then recorded in the Needs document which drives the requirements generation phase.

The content and format of the Needs document is the same as in Prioritized Needs document, which consists of structured and ranked needs. However, the Needs document includes the complete set of needs, which has been refined and validated by the stakeholders (Appendix B.2.13).

#### 4.1.2.18. Need-Problem Traceability

The Mapping Needs to Problems activity (Appendix A.2.3) takes the Needs document as input and generates the Need-Problem Traceability document. This document helps in determining whether all the problem elements have been addressed. In addition, it is used in identifying the impact of changes to the needs.

The Need-Problem Traceability document is usually a traceability diagram, which shows the dependencies between the needs and problem elements. Each need in this document should have a unique identification number and a description.

The format of the Need-Problem Traceability document is either a matrix or a tree (Appendix B.2.14). Traceability trees are easier to manage but one can easily overlook dependencies. On the other hand, matrices are comprehensive, but are difficult to manage because of the complexity. The Need-Problem Traceability document is a useful supplement to the Requirements Traceability document when handling requirements changes.

### *4.1.3. Requirements Capturing Documents*

The Requirements Capturing phase (Appendix A.3) involves the elicitation and analysis of the requirements. This is an iterative phase and is repeated until the complete set of requirements is obtained. In addition, this phase also performs the verification and validation of the elicited sets of requirements. The participation of users is critical to the success of this phase, since most of the activities in this phase are driven by user inputs/feedback. In order to meet the objectives of this phase, it is necessary to conduct requirements elicitation meetings, analyze the requirements, check quality characteristics, and validate the captured requirements.

There are several input documents for this phase – Requirements Engineering Principles, Needs document, Systems Context and Constraints documents, and Stakeholder Profile. These documents provide information which simplifies the task of conducting the activities defined in the Requirements Capturing phase.

This phase produces documents starting with the Unstructured Set of Requirements and ending with the complete Requirements List. The documents evolve incrementally, and most of them are represented in a textual format except for the inclusion of analysis models which depict the requirements graphically.

#### 4.1.3.1. Unstructured Set of Requirements

The Unstructured Set of Requirements is generated by the Requirements Elicitation activity (Appendix A.3), which focuses on meeting with the stakeholders to capture the user requirements. The input documents include System Context and Constraints document, Stakeholder Profile, and the Needs document. During the elicitation activity,

requirements are obtained from the user based on their specified needs. The System Constraints document helps in deriving the non-functional requirements while the Stakeholder Profile is used in identifying participants of the elicitation activity.

Based on the Needs document, requirements are recorded in the Unstructured Set of Requirements. The requirements engineer should ensure that the requirements are well stated. It is recommended that the following guidelines are followed:

- Keep the sentences short and use the active voice. Use proper grammar, spelling, and punctuation.
- Use terms consistently and define them in a glossary or data dictionary.
- Avoid long narrative paragraphs that contain multiple requirements.
- Use “shall” rather than “should/will”.
- Avoid aggregating multiple requirements into a single statement. Conjunctions like “and” and “or” in a requirement suggest that several requirements have been combined. Avoid the use of “and/or” in a requirement statement.

Requirements are often stored in a database in order to facilitate easier management. Several commercial tools are available that transform the information in the database into a structured document. The Unstructured Set of Requirements is represented in a textual format (Appendix B.3.1). In addition, the document may include supplementary information in the form of scenarios and/or diagrams.

#### 4.1.3.2. Models and Refined Requirements

Modeling facilitates understanding and analysis of the requirements. Requirements models are generated during the Local Analysis activity (Appendix A.3), after the requirements have been elicited from the users. The input for requirements modeling is the Unstructured Set of Requirements. Depending on the system being developed, the requirements are modeled in different forms to capture the data flow, system states, and decision logic (Appendix B.3.2). Some of the models are discussed below:

- **Dataflow Diagram** (DFD) – identifies the transformational processes and the flow of data between the processes and the outside world. DFD outlines user tasks

in a graphical way and provides compact specification of the needed data. This model is suited for business applications which are data driven. However, the DFD is not suited to describe user tasks with many variations.

- **State Transition Model** (Finite State Machine) – illustrates how the system responds to internal and external events. Hence they are also referred to as stimulus-response models. These models identify the states of the system when a particular input is given. The state transition model is suitable for real time, complex applications.
- **Data model** (Entity-Relationship model) – specifies the data entities to be stored in the system and the relationships among them. Entity-Relationship (E/R) modeling is a commonly used data model. The E/R model shows data entities, associated attributes, and the relationships among the data entities. Many developers are experienced working with E/R models, and the precise description of data makes the model simple to verify. However, customers often find the E/R diagrams difficult to understand [Lauesen 2002].
- **Decision table/tree** – are widely used for recording decision logic. They specify the actions to be performed given a particular set of conditions. These are excellent tools to choose between several courses of action, and are easy to validate. In addition, they form a balanced picture of the risks and rewards associated with each possible course of action. However, decision trees can become overly complex and large.
- **Data dictionary** – a textual description of data inside and outside the product. This is a supplementary model to the other requirement models. It helps in the consistent use of names in the models. In general, data dictionaries work best with data models. A data dictionary facilitates the validation process as it is easy to comprehend for both developers and users. However, the biggest drawback is that it takes a long time to write, and it is also difficult to decide the level of detail.

Each of these models illustrates different types of information about the requirements. The selection of the models should be based on the type of application being developed. The models created in this activity are used as supplementary information in the SRS.

The models provide a better understanding of the requirements and help in their refinement. Subsequently, the unstructured set of requirements is modified based on the analysis of the models and participation of the users. Except for a few changes, the refined requirements are the same as the unstructured set of requirements.

#### 4.1.3.3. Structured Set of Requirements

The Structured Set of Requirements is generated in the Local Analysis activity, after the requirements are refined based on modeling. The main purpose is to structure the requirements document by grouping them under common categories. In order to do that, the first step is to distinguish between the functional and non-functional requirements. Additionally, the requirements engineer has to determine the subcategories of both functional and non-functional requirements.

The content of the Structured Set of Requirements consists of the requirements arranged according to the functional and non-functional categories. Within the functional category, the requirements are further classified based on the sub-functionalities. The non-functional requirements are grouped into categories such as performance, security, maintenance, and so on. In some situations, it is necessary to organize the requirements using parameters other than functionality. The IEEE recommends classification of requirements based on user classes, modes of operation, stimulus, and so forth [IEEE 1993].

The structured set of requirements generally follows the format prescribed by the IEEE; the template is included in Appendix B.3.3.

#### 4.1.3.4. Prioritized Requirements

Once the requirements are modeled and organized, they are then ranked according to their importance to the users. In addition, the attributes of the requirements are identified and their values are determined. Thus, the Prioritized Requirements document consists of ranked requirements with associated attributes. The input to the prioritization step in the Local Analysis activity is the Structured Set of Requirements.

The Prioritized Requirements document should include the attributes and values for the requirements elicited from the users. Common attributes[16] that should be considered are – risk factors, effort, rationale, and user importance. A suitable template for the requirements showing their attributes is included in Appendix B.3.4. Once the attributes are identified, the Structured Set of Requirements is prioritized based on the user importance. Generally this is achieved by grouping the requirements into three priority categories – high, medium, and low. The structured set of requirements is then re-organized such that the high importance functionalities are at the top and the others are below. This ranking is also done for the requirements within the sub functionalities so that it is easy to identify the high priority requirements within a particular system feature. Updating and prioritizing the requirements in a document is cumbersome. Hence, a database is commonly used for the storage of requirements; requirements tools can be used to prepare the output document in a suitable format.

The format of the Prioritized Requirements is similar to the Structured Set of Requirements with the difference being that the requirements are now ranked and have associated attributes (Appendix B.3.4).

#### 4.1.3.5. List of Updated Requirements

The List of Updated Requirements is generated by the Requirements Evaluation activity (Appendix A.3), which performs verification and validation on the elicited requirements. The input to this activity is the Prioritized Requirements document. If the users need to refine the requirements, the changes are documented and are passed to the local analysis activity.

The updated requirements document includes the following information for each modified requirement:

- The original requirement
- The modified requirement
- Justification for the change

[16] See Section 3.3.3.

- Author of the change
- Date of modification
- Impact of the change.

While documenting the List of Updated Requirements, the requirements engineer should follow guidelines recommended for specifying requirements[17]. Requirements updates are easier to manage when the requirements are stored on a database.

#### 4.1.3.6. List of Unaddressed Needs

On completion of the Requirements Evaluation activity, the requirements engineer determines whether the requirements are complete. In order to determine the completeness of the requirements it is necessary to check whether all needs have been addressed by the requirements. If some of the needs have not been discussed with the stakeholders, the requirements engineer records these needs in the List of Unaddressed Needs document.

This document has a format similar to the Needs document. It is a subset of the Needs document and is used in the Requirements Elicitation activity.

#### 4.1.3.7. Requirements

The Requirements document is generated when the complete set of requirements has been elicited from the users. This document is composed of requirements which are organized [18] based on functionality, user class, modes of operation, or stimulus. Furthermore, the requirements are ranked[19] to simplify the task of identifying the high priority requirements. The format of the Requirements document is the same as the Prioritized requirements document, with the only difference being that the requirements are complete, verified, and validated (Appendix B.3.6).

[17] See Section 4.1.3.1.
[18] See Section 4.1.3.3.
[19] See Section 4.1.3.4.

### *4.1.4. Global Analysis Documents*

The Global Analysis phase (Appendix A.4) involves evaluating the complete set of requirements from different perspectives. It entails analyzing the requirements for risk, price, cost, schedule, and feasibility. In addition, this phase identifies conflicts among the requirements and resolves them.

This phase has the following input documents – Requirements, System Context and Constraints, Organizational Standards Rules and Protocols, and External Influences document. The Requirements document is generated during the Requirements Capturing phase; the Context and Constraints documents are created in the Problem Synthesis phase. The Organizational Standards and External Influences documents describe the business, market and system environment constraints, and help in the evaluation of the requirements.

Several documents are created in this phase, and correspond to the different evaluation parameters – cost, risk, schedule, feasibility, and price. In addition, this phase incrementally generates a list of requirements conflicts, which are resolved during the Conflict Resolution activity. Each of these documents is described in the subsequent sections.

#### 4.1.4.1. Risk Analysis Document

The Risk Analysis document estimates the risk associated with the implementation of the requirements. Several risk factors need to be evaluated in order to obtain the estimate. In addition, it is necessary to determine the loss incurred if a particular risk materializes. The Risk Analysis document also identifies the high risk requirements that need to be discussed with the customer/user. Furthermore, the document also includes the recommendations of the requirements engineer to help mitigate the high risks.

The Risk Analysis document should include the evaluation of the requirements for the various risk factors. For each component (group of requirements), the document should specify the probability of occurrence that can be attributed to each risk factor. The probability of loss that would result from the occurrence of each risk factor is also included. In addition, the Risk Analysis document should provide the risk factor

criticality, which is the product of the probability of loss and occurrence. The risk estimate is the sum of the criticalities for all the risk factors. The requirements which are above the critical risk estimate are flagged as high risk and are included in a separate section. Furthermore, the requirements engineer can provide his/her own recommendations to reduce the risks involved. Thus, the Risk Analysis document is similar to the Requirements document but with additional risk attributes, and sections for high risk requirements and recommendations.

The Risk Analysis document is a management document and is written in a formal, structured textual format (Appendix B.4.1).

#### 4.1.4.2. Estimated Cost and Schedule

The Cost and Schedule Estimate document determines the cost and time needed to implement the requirements. This document is critical to management as they impact the planning, budgeting, and other managerial decisions. The input documents to the Cost and Schedule Estimation activity (Appendix A.4) is the Requirements document, Context and Constraints document.

This document should include the cost estimation calculations as well as those for determining the schedule estimate. Additionally, cost and schedule intensive requirements are listed in a separate section of the document and are used for later discussion with the management and the customer.

The Cost and Schedule Estimate document is also a management document, and it aids in the allocation of resources. This document is written in a textual format with diagrams representing the schedule (Appendix B.4.2).

#### 4.1.4.3. Market Survey Document

The Market Survey document specifies market information and opinions of the users. This document is generated in the Market Price Analysis activity (Appendix A.4), and the output is used to estimate the product price.

The Market Survey document is usually created by examining the market through questionnaires. This document should include the following information:

- Number of customers and competitors in the market
- Prices of similar/competitor products
- Intensity of demand
- Quality of products in the market
- Important/desirable product features.

This document is written in a textual format in a natural language.

#### 4.1.4.4. Market Value Document

The Market Survey document specifies the product price that is reasonable and fair considering the market demand. It also lists the product features that can be left out of development because they do not add value to the product. The input for Market Analysis is the Requirements and the Market Survey document. The market information helps to decide whether the price is reasonable for a particular product feature.

The Market Value document should record the statistics involved in obtaining the price estimate. Thus, for each product feature, the document should specify the relevant market information such as the competitor's price, the user demand, and so forth. While comparing prices for fairness, it is necessary that the prices be adjusted for inflation/deflation. If the price is not reasonable, the requirements engineer should justify this decision. In addition, this document includes a section that specifies features which do not add value to the product and which can be dropped from the current development cycle (Appendix B.4.4).

This document uses a textual format interspersed with calculations.

#### 4.1.4.5. System Feasibility Document

The System Feasibility document specifies whether the product is practical and profitable to develop. In order to make this decision, the analyst needs to evaluate the operational, technical, schedule, and economic feasibility of the product. The System Feasibility document is generated during the Feasibility Analysis activity (Appendix A.4), which takes all the previously generated Global Analysis documents as input.

The System Feasibility document includes a separate section for each of the feasibility evaluations – operational, technical, economic, and schedule.

- **Operational feasibility** – documents the ease of use and the effectiveness of the system. It should include information about the managerial support, training required, workforce reduction, and effects on customers and users.
- **Technical feasibility** – assesses the development team's understanding of the proposed system hardware, software, and the operating environment. It determines whether the lack of technical expertise affects the schedule and cost of the project.
- **Schedule feasibility** – determines if the schedule estimate is reasonable and meets customer's expectations.
- **Economic feasibility** – specifies whether the product is economically viable. It calculates the number of years after which the product becomes profitable. Several methods are used for this purpose, and the commonly used method among them is the Net Present Value technique. Table 4.3 illustrates the format of the information generated by this method. Cash flow is the revenue earned by the product in a particular year. The inflation rate considers the inflation/deflation rate to adjust the revenue earned. Present values give the actual money procured in a particular year.

| Year | Cash Flow | Inflation Rate | Present Value |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| … | | | |
| Net Present Value | | | |

*Table 4.3 – Net Present Value*

The last section in the Feasibility document specifies the overall profitability of the product by considering the results of the feasibility evaluations. This section lists the

positive and negative results of the project in a table and tallies them. A decision is then made as to whether the project is feasible or not; the reasoning for this decision is also included.

This document is presented in a textual format consisting of several tables corresponding to the feasibility calculations (Appendix B.4.5).

#### 4.1.4.6. Set of Conflicting Requirements

Set of Conflicting Requirements document is created during the Feasibility Analysis activity. The activities in the Global Analysis phase – Risk, Cost and Schedule, Market Price, and Feasibility Analysis – all detect conflicts in the requirements if they exist. However, all the conflicts are put together in a single document during the Feasibility Analysis activity.

This document includes the requirements which conflict with another. In addition, the Set of Conflicting Requirements document also specifies the requirements that do not satisfy the customer specification of schedule, cost, and so on. This document uses the stakeholders' vernacular to explain the conflicts and is expressed in a textual format (Appendix B.4.6). If additional information about the conflicting requirements is required, the Requirements document can be referenced.

#### 4.1.4.7. Conflict-Free Requirements

This document consists of the complete set of requirements that are free from conflicts. This document is generated by the Conflict Resolution activity (Appendix A.4), whose input is the Set of Conflicting Requirements and the Requirements documents. Once the conflicts are negotiated, the resolved requirements are incorporated in to the Requirements document to produce the Conflict-Free Requirements document.

The content and format of the Conflict-Free Requirements document is the same as the Requirements document; the only difference being that the requirements are conflict-free (Appendix B.4.7). This document becomes a component of the SRS and follows the prescribed presentation standards.

### *4.1.5. Requirements Specification Documents*

The Requirements Specification phase/activity (Appendix A.5) involves the generation of the Requirements Specification. This document is important because it is the basis for communication among the stakeholders. Furthermore, it serves as a contract between the customer and developer and is a reference in the later phases of the development life cycle. The objective of this activity is to compile the collected information in a comprehendible and clear manner.

This activity has several input documents including Vision document, Conflict-Free Requirements, System Context and Constraints, and Problem Statement. Each of these documents contributes sections of the Requirements Specification.

#### 4.1.5.1. Non-Validated Requirements Specification

The output of the Specification activity is the Non-Validated Requirements Specification, whose template is provided in Appendix B.5.1. The Requirements Specification is an elaborate document; much research has been conducted in defining its content. Generally the Requirements Specification consists of the following sections:

- **Introduction** – presents overview of the Requirements Specification and describes its organization.
    - Purpose – identifies the proposed system.
    - Document Conventions – describes any document standards followed.
    - Intended Audience – lists the readers to whom the requirements specification is directed; for example, customers, developers, managers, and so forth.
    - Product Scope – provides a brief description of the software, its purpose, benefits, and objectives.
    - References – lists any documents or other resources to which the Requirements Specification refers.

- **Overall Description** – presents a high-level overview of the product, its users, environment, assumptions, and dependencies.
    - Current System or Situation – provides an overview about the current system/situation.
    - Justification for Changes – presents the rationale for system changes and the benefits of incorporating the changes.
    - Product Perspective – describes the context of the proposed system.
    - Product Functions – summarizes the major functions the system must perform.
    - User Classes and Characteristics – identify intended users and specify their characteristics.
    - Operating Environment – describes the environment in which the software will operate.
    - Design and Implementation Constraints – restrictions imposed by other standards or hardware/software limitations.
    - Assumptions and Dependencies – these factors are not design constraints on the software, but rather any changes to them that can affect the requirements in the Requirements Specification.
- **External Interface Requirements** – specifies requirements that help ensure the new product connects properly to external components.
    - User Interfaces – states the software components for which a user interface is needed.
    - Hardware Interfaces – describes the characteristics of each interface between the software and hardware components of the system.
    - Software Interfaces – describes the connections between the system and other external software components.

- **System Features** – enumerates all features that must be implemented in the system.
    - Name – states the name of the feature in a few words.
    - Description and Priority – provides a short description of the feature and indicates whether it is of high, medium, or low priority.
    - Functional Requirement – specifies the functional requirements associated with this feature.
- **Non-Functional Requirements** – lists requirements pertaining to performance, safety, security, and so on.
    - Performance – specifies static or dynamic numerical requirements placed on the software.
    - Reliability – lists factors required to establish the reliability of the software system.
    - Security – describes factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure.
    - Maintainability – specifies attributes of the software that relate to the ease of maintenance of the software itself.
    - Portability – lists attributes of the software that relate to the ease of porting the software to other host machines or operating system.
- **Other Requirements** – defines any other requirements that are not covered elsewhere in the Requirements Specification, such as internationalization requirements or legal requirements.
- **Appendix**
    - Glossary – defines all the terms, necessary for the reader, to properly interpret the requirements specification.
    - To-Be-Determined List – compiles a numbered list of the TBD (to be determined) reference that remain in the requirements specification.

- Scenarios and Models – provides supplementary information to facilitate the understanding of the requirements.

The Requirements Specification compiles all the essential information generated from the preceding activities of the requirements generation process. The document is usually written in a textual format, and diagrams and scenarios can supplement the understanding of the requirements.

### *4.1.6. SRS Evaluation Documents*

The SRS Verification and Validation phase (Appendix A.6) performs the final verification, validation, and traceability checks on the Requirements Specification. It involves evaluating the requirements for various quality attributes, conducting meetings with customers/users to obtain feedback, and analyzing the relationships of requirements. In addition, the requirements engineer decides if the requirements generation is complete or another iteration of the process is needed.

The input to this phase is the Non-Validated Requirements Specification generated during the Specification phase. The Requirements Verification and Validation phase produces several documents which help in the refinement of the Requirements Specification. The Requirements Traceability document is also generated and used by the requirements engineer to determine the impact of changes.

#### 4.1.6.1. Verified Requirements Specification

The Verified Requirements Specification document is the list of requirements that have been evaluated for the various quality attributes, such as completeness, consistency, verifiability, correctness, modifiability, and so on. This ensures that the Requirements Specification is unambiguous, complete, and precise.

This content and format of the document is similar to the Requirements Specification document; the difference is that the Verified Requirements Specification document includes a section which lists the requirements that do not adhere to the quality characteristics (Appendix B.6.1). Furthermore, this document may include some

recommendations for correcting the requirements so that they become compliant with the quality attributes.

#### 4.1.6.2. Requirements Traceability Document

The Requirements Traceability document is generated by the Traceability Analysis activity (Appendix A.6). By depicting the traceability among requirements as well as to the needs, this document helps in identifying relationships/dependencies among the requirements and determining the rationale for implementing the requirements. Therefore, the Requirements Traceability document facilitates the change management process in determining the impact of changes to the requirements.

This document is usually a diagram depicting the dependencies of requirements and needs. The format of the traceability document can be either tabular as in a matrix or in the form of a tree (Appendix B.6.2). Traceability matrix is comprehensive and is visual, but the drawback is that it can become complicated and difficult to handle. Traceability tree overcomes the drawbacks of the matrix, but it is not visual as the matrix. The traceability document is sometimes attached as an appendix to the SRS.

#### 4.1.6.3. Requirements Change Request

Requirements Change Requests are generated by the Customer Validation activity (Appendix A.6), which determines if the requirements meet the customer intent. If the customer's needs and the requirements do not match, the necessary changes are documented in the Requirements Change Request document.

The content of the Change Request can vary among organizations. In this section, the general information in a change request document is presented.

- Change request number – used for tracking purposes
- Title – descriptive title for the modification
- Source – author of the modification
- Date – date the change is requested
- Original requirement – requirement that needs to be changed

- Requested change – brief explanation of the necessary changes
- Reason for change – explanation of why the changes are necessary
- Impact of the change – Implications of the change on the system
- Consequences if not approved – description of the consequences if the change request is not approved.

The Change Request document is written in a natural language using a textual or tabular format. It is necessary that the change requests be clear and unambiguous to prevent an erroneous correction of the requirements.

#### 4.1.6.4. Updated Requirements Documents

The Change Requests are analyzed, and the requirements documents are updated according to the changes. The updated documents should ensure that the changed requirements satisfy the customer intent and are consistent with other requirements.

Based on the Change Requests, the Requirements Specification is modified; this also affects the Requirements Traceability document too. If a new feature is added, the global analysis documents may have to be modified to include the evaluation of the new feature/functionality. While updating the requirements and related documents, it is necessary to maintain the consistency of the requirements.

Updating requirements documents is simplified if the requirements are manipulated through tools which support a requirements database.

#### 4.1.6.5. Software Requirements Specification

The Software Requirements Specification (SRS) is the final output of the requirements generation process. This document consists of all the information[20] recommended for a SRS and also adheres to quality attributes. Furthermore, the SRS should meet the customer's needs.

The SRS content and format should comply with the standards prescribed by organizations such as IEEE [IEEE 1993]. The content and format of the SRS is similar to

[20] See Section 4.1.5.1.

the Non-Validated Requirements Specification document, with the only difference being that it is verified for quality attributes and validated by the customers (Appendix B.6.4).

## 4.2. Transformation of the Requirements Documents

The previous sections describe the documents' objective, content, possible formats, and its synchronization with related activities. This section discusses how information is passed through documents during the requirements generation process (Figure 4.3).

The first phase in the requirements generation process is the Conceptual Overview phase which captures the customer's perspective of the problem and the proposed system. This phase can have following inputs – HLR, ConOps, and Ideas. While HLR includes high-level requirements, ConOps describes system features in the user's vernacular. Ideas refer to the user's perception of the problem and the solution. These documents are processed by the Initial Customer Meeting activity, which generates the document – Customer Perspective of the Problem. This document records the customer's viewpoints of the problem, solution, its rationale and benefits. The information elicited from the customer is then organized in the Vision document, which highlights problems and justifies the development of the proposed system.

The next phase is the Problem Synthesis phase, which begins with the Requirements Engineer Education. This activity produces the Domain Model that illustrates key concepts or business entities and their relationships.

The Domain Model is passed to Problem Analysis, which is a sub-phase of Problem Synthesis. The Domain Model along with the Vision document, HLR and ConOps is used by the Problem Identification activity to generate the Problem Statement and Stakeholder Profile. The Problem Statement describes the problem, affected stakeholders, and the benefits of resolving the problem. On the other hand, the Stakeholder Profile includes the stakeholders' names, contact information, their positions and departments.

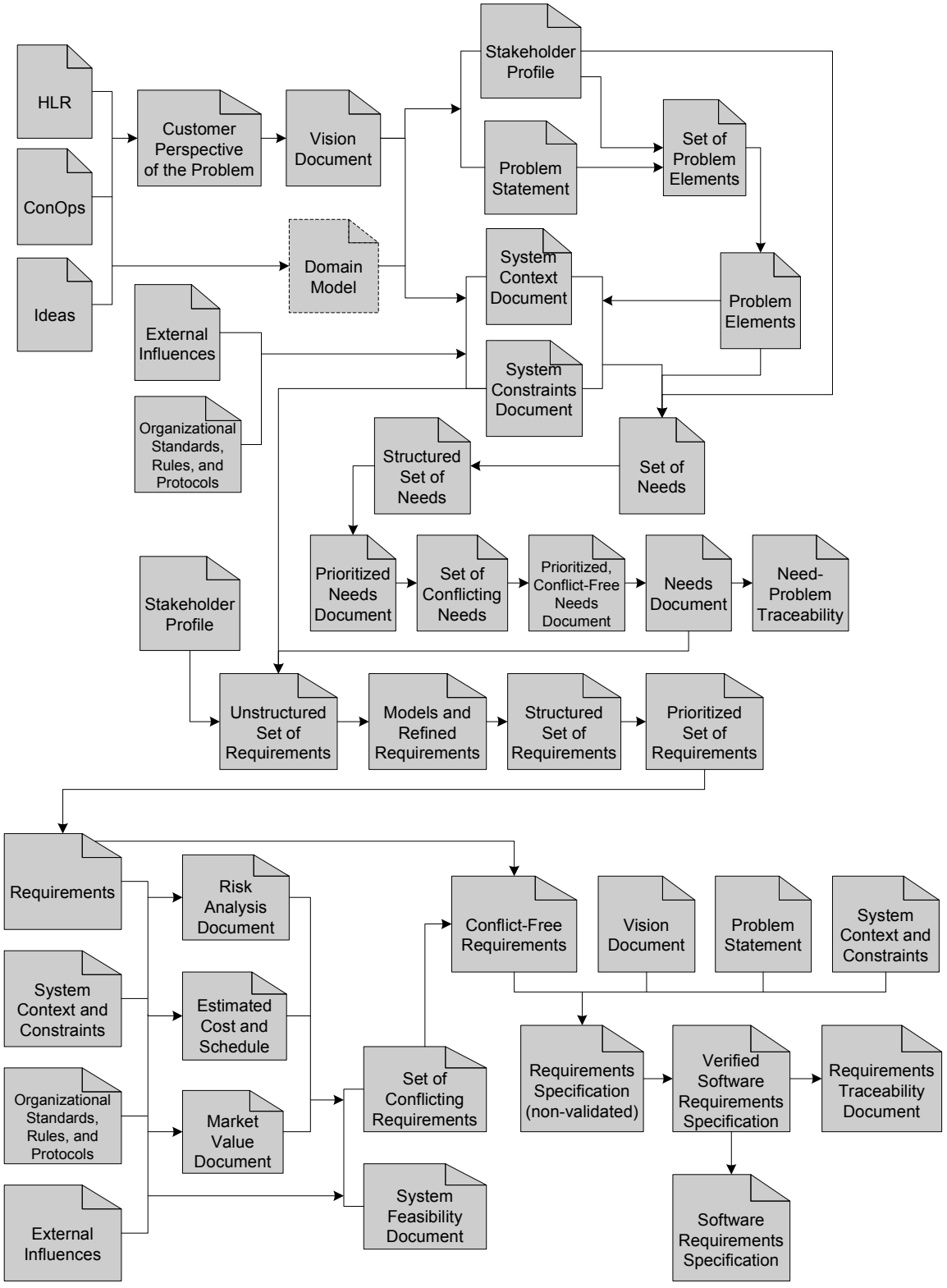



*Figure 4.3 – Transformation of Requirements Documents*

Both of these documents are inputs to the Problem Decomposition activity, which identifies the sub-problems of a particular high-level, complex problem. The output of this activity is the Set of Problem Elements. The Set of Problem Elements is evaluated by the Problem Elements Analysis activity, which generates the Problem Elements document its output. This document is similar in content to the Set of Problem Elements document, but it includes a complete set of problem elements that are ranked. The Domain Model, Vision document, and Problem Elements are inputs to the Context and Constraints analysis activity, which produces the System Context and Constraints documents. The System Context document determines the boundary of the system and describes the data exchanges between the system and its environment. The System Constraints document includes different types of system constraints and their rationale.

Needs Generation is the next sub-phase in the Problem Synthesis phase after Problem Analysis. Its inputs are Problem Elements, System Context and Constraints, and Stakeholder Profile documents. This phase begins with the Needs Elicitation activity, which generates the Set of Needs document. This document consists of user needs to overcome the problem elements. The needs are then integrated and organized according to high-level system features during the Needs Analysis activity. The needs are then ranked either relative to one another or according to priority groups to generate the Prioritized Needs document. The Needs Analysis activity also outputs the Set of Conflicting Needs. This document describes conflicts in the Needs document and the reasons for the conflicts. The conflicts are resolved through negotiation in the Needs Conflict Resolution activity to generate the Prioritized Conflict-Free Needs document. The Conflict-Free Needs are then evaluated. As a result of the evaluation, recommended changes are recorded in the List of Updated Needs. When the changes are updated, the complete set of Needs is generated. The Need-Problem Traceability document is the last document generated in the Needs Generation phase and is usually depicted by a traceability diagram, which shows the dependencies of the needs and problem elements

Once the Needs document is obtained, the Requirements Capturing phase begins, and takes as its input the System Context and Constraints, Stakeholder Profile, and the Needs document. The Needs document drives the Requirements Elicitation activity which produces the Unstructured Set of Requirements. These requirements are then modeled

and refined during the Local Analysis activity. Depending on the system being developed, the requirements are modeled to capture the data flow, system states, and decision logic, and so on. To facilitate a better understanding, the requirements are organized into functional and non-functional categories to produce the Structured Set of Requirements. These categories are further broken down into sub-categories. Finally, the Local Analysis activity ranks the requirements and produces the Prioritized Requirements document. This document includes values of requirement attributes, such as risk factors, effort, and user importance. The Prioritized Requirements document is analyzed by the Requirements Evaluation activity. The evaluation results are recorded in the List of Updated Requirements. This document includes the modified requirements, reason for change, author of the change, and so on. On completion of the Requirements Capturing phase, the Requirements document is generated, and includes the verified and validated requirements.

The next phase is Global Analysis, which has the following inputs – Requirements document, System Context and Constraints, Organizational Standards, Rules and Protocols. This phase produces the Risk Analysis document, which includes the evaluation of requirements for the various risk factors. In addition, high risk requirements are identified and risk mitigation measures are suggested in this document. The Estimated Cost and Schedule document is also generated by the Global Analysis phase and is composed of cost/schedule estimation calculations and diagrams, as well as a section for cost and time intensive requirements. Market Analysis activity produces the Market Value document, which includes the reasonable price for the system, price estimation calculations. The Feasibility Analysis activity in the Global Analysis phase generates two documents – System Feasibility document and Set of Conflicting Requirements. The System Feasibility document records the practicality of the system and addresses operational, technical, schedule, and economical perspectives. This document also includes the reasoning for the decision as to whether the system is feasible or not. The Set of Conflicting Requirements includes the requirements which conflict with each other, and also contains requirements that do not satisfy the customer specification of schedule, cost, and so on. On negotiating the conflicts, the Conflict-Free Requirements document is generated.

The Conflict-Free Requirements are then organized according to the SRS standards to produce the Non-Validated Requirements Specification. This document includes the overall description of the system and the detailed functional and non-functional requirements.

SRS Evaluation is the next phase; it performs the final evaluation on the Requirements Specification. On verifying the specification, the Verified Requirements Specification document is generated, and includes an additional section of the requirements that do not adhere to the quality characteristics. The verified requirements are then linked in a Requirements Traceability document, which depicts traceability among requirements as well as to the needs. The Requirements Specification is validated by the customer, and based on the feedback Requirements Change Request document is created. This document includes the requested changes, reason for changes, and consequences if changes are not approved. Based on the Change Request document, the requirements documents are modified to reflect the changes. If the customer approves the requirements, the Customer Validation activity produces the Software Requirements Specification (SRS). The SRS is the final document of the requirements generation process, which is baselined and signed by the customer.

**Summary:** This chapter discusses the content and format of the documents produced during the requirements generation process. In addition, the evolution of the requirements documents throughout the requirements engineering process is described. In Chapter 5, we provide a brief summary of this research and its contributions to the field of requirements engineering. Furthermore, the opportunities of extending this research work are also discussed.

# CHAPTER 5. SUMMARY AND FUTURE WORK

## 5. Summary and Future Work

This research identifies the intermediate requirements documents and describes how these documents evolve in terms of content and format during the requirements generation process. Chapter 1 describes the importance of this research and identifies the issues involved in this work. In Chapter 2, we present the background and related work for this research. In addition, the inadequacies of the current requirements engineering models are discussed, and their lack of guidance on the intermediate documents is highlighted. Chapter 3 describes the refinement and decomposition of the RGM, and also presents the objectives of its identified activities. Chapter 4 focuses on the evolution of requirements documents during the requirements generation process, and describes the content and format of these documents. In this chapter, we present the summary of this research and highlight the contributions of this work to the requirements engineering field. Furthermore, it also briefly describes potential extensions to this research.

## 5.1. Summary

This research focuses on evolving forms of requirements documents throughout the requirements engineering process and leading to a well-defined SRS. Most of the current requirements engineering models either address only parts of the requirements generation process or, if they do include the full process, do so at an insufficient level of detail. In addition, those models fail to identify the *intermediate* documents produced during the requirements generation process. Unfortunately, the above inadequacies often lead to ill-defined intermediate requirements documents, and subsequently, a less than desirable SRS.

In order to address these problems, we considered the following set of issues:

- Insufficient decomposition of the activities in the requirements generation model

- Implicit activity objectives
- Inadequate identification of the intermediate document characteristics
- Lack of synchronization between documents and activity objectives
- Lack of understanding about the evolution of requirements.

The RGM is chosen as the basis for this research because it includes all the major requirements engineering phases and is flexible to changes. Using "separation of concerns" as a guide, we conducted a literature survey on the current requirements engineering models, identified the positive features of those models, and incorporated them into the RGM model. On completion of designing the model, the refined RGM includes the following phases:

- **Conceptual Overview** – captures the overview of the customer's viewpoint about the problem and the proposed system.
- **Problem Synthesis** – identifies the real problem and the root cause behind the problem, decomposes the problems into sub-components (problem elements), and determines the user needs for those problem elements.
- **Requirements Capturing** – elicits, analyzes, and evaluates sets of requirements iteratively until all the requirements are obtained and recorded.
- **Global Analysis** – evaluates the complete set of requirements from different perspectives. This phase also includes the negotiation of requirements conflicts.
- **Requirements Specification** – the information captured from the previous phases are organized according to the prescribed specification standards.
- **SRS Evaluation** – performs verification and validation on the requirements specification produced in the preceding phase.

These phases are decomposed into detailed activities to reflect an appropriate level of abstraction, so that we can accurately represent the intermediate development of the requirements documents. In addition to identifying the activities, the objectives of the activities are also determined.

Once the activities and their objectives are identified, the content and format of the documents produced by each activity are formulated. In order to achieve this goal, documents proposed by the researchers, and the ones used in the industry, are analyzed to obtain insights into requirements representations. The documents are synchronized with the activities to ensure that the input/output documents support the activity objectives. The evolution of the requirements, in terms of content and format, is presented as requirements documents pass through successive activities within the requirements engineering process.

## 5.2. Contributions

The intent of the refined RGM model, with the support from related artifacts, is to present a clear picture of requirements generation and to provide guidance for the requirements engineer. The detailed contributions of this research to the requirements engineering field are discussed below:

- **A refined requirements engineering model** – The refined RGM model provides a comprehensive view of requirements engineering process, consisting of six phases and twenty six activities. The decomposition of the RGM into activities is shown in Table 5.1.

| Phase | Number of Activities | Activity Name | |
|---|---|---|---|
| **Conceptual Overview** | 2 | Initial Customer Meeting | |
| | | Documenting System Characteristics | |
| **Problem Synthesis** | 10 | Requirements Engineer Education | |
| | | Problem Analysis | Problem Identification |
| | | | Problem Decomposition |
| | | | Problem Elements Analysis |
| | | | Context and Constraints Analysis |

| Phase | Number of Activities | Activity Name | |
|---|---|---|---|
| **Problem Synthesis (contd.)** | 10 | Needs Generation | Needs Elicitation |
| | | | Needs Analysis |
| | | | Needs Conflict Resolution |
| | | | Needs Evaluation |
| | | | Mapping Needs to Problems |
| **Requirements Capturing** | 4 | Customer Education | |
| | | Requirements Elicitation | |
| | | Local Analysis | |
| | | Requirements Evaluation | |
| **Global Analysis** | 5 | Risk Analysis | |
| | | Cost and Schedule Estimation | |
| | | Market Price Analysis | |
| | | Feasibility Analysis | |
| | | Conflict Resolution | |
| **Requirements Specification** | 1 | Requirements Specification | |
| **SRS Evaluation** | 4 | Verifying Quality Attributes | |
| | | Requirements Traceability | |
| | | Customer Validation | |
| | | Updating Requirements Documents | |

*Table 5.1 – Activities identified in the Refined RGM*

- **Identification of activity objectives** – The current requirement engineering models often define only the high-level activity objectives, and imply the lower level ones. This research explicitly identifies the objective of each activity in the refined RGM. The identification of the activity objectives is important, because the documents are mapped to the requirements generation process based on the activity objectives.

- **Synchronization of document and activity objectives** – A drawback of the current requirements engineering models is that the intermediate and final embodiments of the requirements lack the necessary characteristics needed for comprehension and communication. In this research, we identify the intermediate documents needed to produce a complete and clear requirements specification. That is, those documents are synchronized with activity objectives through an explicit explication of content and format.
- **Determining the evolution of requirements** – Finally, the current models fail to depict the evolution of requirements during the requirements generation process. This research addresses this inadequacy by formally integrating intermediate (and final) requirements documents throughout the requirements generation process, starting with the identification of the problem and ending with the generation of the SRS. Table 5.2 identifies the input *and output documents* employed in the refined RGM.

| Phase/Activity Name | Number of Inputs | Output Document Name |
|---|---|---|
| **Conceptual Overview** | | |
| Initial Customer Meeting | 3 | Customer Perspective of the Problem |
| Documenting System Characteristics | 4 | Vision Document |
| **Problem Synthesis** | | |
| Requirements Engineer Education | 4 | Domain Model |
| Problem Identification | 5 | Problem Statement |
| | | Stakeholder Profile |
| Problem Decomposition | 2 | Set of Problem Elements |
| Problem Elements Analysis | 1 | Problem Elements |
| Context and Constraints Analysis | 5 | System Context |
| | | System Constraints |
| Needs Elicitation | 4 | Set of Needs |

| Phase/Activity Name | Number of Inputs | Output Document Name |
|---|---|---|
| Needs Analysis | 1 | Prioritized Needs |
| | | Set of Conflicted Needs |
| Needs Conflict Resolution | 2 | Prioritized, Conflict-Free Needs |
| Needs Evaluation | 1 | Needs Document |
| Mapping Needs to Problems | 2 | Need-Problem Traceability |
| **Requirements Capturing** | | |
| Customer Education | 1 | N/A (Customer Knowledge of Requirements Engineering) |
| Requirements Elicitation | 4 | Unstructured Set of Requirements |
| Local Analysis | 1 | Prioritized Requirements |
| Requirements Evaluation | 1 | Requirements |
| **Global Analysis** | | |
| Risk Analysis | 4 | Risk Analysis Document |
| Cost and Schedule Estimation | 4 | Estimated Cost and Schedule |
| Market Price Analysis | 5 | Market Value Document |
| Feasibility Analysis | 7 | System Feasibility |
| | | Set of Conflicted Requirements |
| Conflict Resolution | 2 | Conflict-Free Requirements |
| **Requirements Specification** | 5 | Requirements Specification (non-validated) |
| **SRS Evaluation** | | |
| Verifying Quality Attributes | 1 | Verified Requirements Specification |
| Requirements Traceability | 2 | Requirements Traceability |
| Updating Requirements Documents | 1 | Updated Requirements Documents |
| Customer Validation | 1 | SRS |

*Table 5.2 – Input/Output Documents Identified for Activities in the Refined RGM*

## 5.3. Future Work

This thesis provides several opportunities for extended research. In this section we discuss some important extensions to this work.

- **Empirical evaluation of the refined RGM –** We conjecture that the refined RGM model enhances the requirements engineering process by generating a more complete and correct specification. However, we need to evaluate the model in a real world setting in order to determine the drawbacks, if any, of the model. Furthermore, this study would be useful in proving the effectiveness of the model. To perform the evaluation, it is necessary to identify efforts which are representative of the real world projects, and apply the process defined by the refined RGM
- **Extending the model to other software development paradigms –** The proposed requirements model is based on the waterfall model, which requires the generation of complete set of requirements before commencing the design phase. However, this model could be adapted to include the characteristics of other development paradigms such as incremental, evolutionary, and so forth. To adapt this model, it is important to determine how the various phases of the development cycle overlap with the requirements phase in the different paradigms.
- **Implementation of a management tool:** In this research, we identified the activities and documents of the refined RGM. To help optimize the requirements generation cost and quality, we need to provide guidance to the requirements engineer relative to the activities and associated documents. One approach to achieve this "optimization" is to provide a management tool that offers document templates for intermediate and final representations of requirements at the appropriate times during the requirements generation process.
- **Change management and version control documents**: This research focuses on producing the SRS as the final product; it ignores, somewhat, the issues of change management and version control. Hence, a natural extension to this research

would be to provide the ability to manage corrections for existing requirements, and to capture new requirements as they are determined in later phases of the software development. To achieve this capability, additional research is needed to determine the change management process model that can analyze changes and incrementally incorporate them into the sets of requirements documents.

- **Incorporating the usability engineering process** – Currently, the refined RGM model focuses only on the software engineering perspective of product development. As a part of the future work, the model could be adapted to incorporate both requirements engineering and usability engineering approaches. For this work, the principles and characteristics of the usability engineering process need to be considered. Identifying the differences between the approaches is also critical in synchronizing these models.

## References


[Aksit 2001] Aksit, M., Tekinerdogan, B., and Bergmans, L., "The Six Concerns for Separation of Concerns", *in Proceedings of ECOOP 2001 Workshop on Advanced Separation of Concerns,* Budapest, Hungary, June 18-22, 2001.

[Arthur 1999] Arthur, J., and Gröner, M., "An Operational Model Supporting The Generation of Requirements That Capture Customer Intent", *Annual Pacific Northwest Software Quality Conference*, Portland, OR, 1999.

[Bailin 1997] Bailin, S., "Object-Oriented Requirements Analysis", *Software Requirements Engineering* by Thayer. R., 1997.

[Basili 1981] Basili, V., and Weiss, D., "Evaluation of a Software Requirements Document by Analysis of Change Data", *IEEE Conference on Software Engineering*, 1981, pp. 314-23.

[Beck 1999] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[Bell 1976] Bell T. E., Thayer, T., "Software requirements: Are they really a problem?", *Proceedings of the 2nd international conference on Software engineering,* 1976.

[Beohm 1988] Boehm, B., "A Spiral Model for Software Development and Enhancement", *Computer*, vol. 21, pp. 61-72, 1988.

[Beohm 1998] Boehm, B., "Using the Win-Win Spiral Model: A Case Study", *IEEE Computer*, July 1998.

[Brackett 1990] Brackett J., "Software Requirements", *SEI Curriculum Module SEI-CM-19-1.2,* January 1990.

[Brooks 1987] Brooks F.P., Jr., 1987, "No Silver Bullet: Essence and Accidents of Software Engineering", *IEEE Computer*, v.20 n.4, April 1987.

[Brooks 1995] Brooks, F., *The Mythical Man-Month*, Addison-Wesley, 20th aniversary edition edition, 1995.

[Bubenko 1994] Bubenko, J., Rolland. C, Loucopoulos, P., and DeAntonellis, V., "Facilitating fuzzy to formal requirements modeling", *In Proceedings of the IEEE First International Conference on Requirements Engineering (ICRE94),* 1994.

[Carroll 1995] Carroll, J., *Scenario-Based Design: Envisioning Work and Technology in System Development*, Wiley, 1995.

[Celko 1983] Celko, J., Davis, J., and Mitchell, J., "A Demonstration of Three Requirements Language Systems", *ACM SIGPLAN Notices*, January 1983.

[Christel 1992] Christel, M. and Kang, K., "Issues in Requirements Elicitation", *Technical Report CMU/SEI-92-TR-12 ESC-TR-92-012,* 1992.

[Chung 1995] Chung, L., Nixon, B., and Yu, E., "Using non-functional requirements to systematically support change", *In Proceedings of the second IEEE International Symposium on Requirements Engineering (RE95),* 1995.

[Chung 1999] Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., *Non-Functional Requirements in Software Engineering*, Kluwer Publishing, 1999.

[Cooper 1996] Cooper, A., "Goal-Directed Software Design", *Dr. Dobb's Journal*, September 1996.

[Constantine 1989] Constantine, L., "Object-Oriented and Structured Methods: Toward Integration", *American Programmer,* August 1989.

[Cysneiros 1999] Cysneiros, L., and Leite, J., "Integrating non-functional requirements into data modeling", *Requirements Engineering, Proceedings IEEE International Symposium,* 7-11 June 1999.

[Cysneiros 2002] Cysneiros, L. and do Prado Leite, J., "Non-functional requirements: from elicitation to modeling languages", *ICSE Proceedings of the 24rd International Conference,* May 2002.

[Davis 1990] Davis, A., *Software Requirements: Analysis & Specifications,* Prentice-Hall, 1990.

[Davis 1993] Davis, A., *Software Requirements: Objects, Functions, and States,* Prentice-Hall, 1993.

[Davis 1993A] Davis, A., "Status Report: Requirements Engineering", *IEEE Software,* 1993.

[Davis 1999] Davis A., Fairley, R., and Yourdon, E., "Software Product Planning", Omni-Vista White Paper #99-002, October 1999.

[DeMarco 1978] DeMarco, T., *Structured Analysis and System Specification,* Yourdon Press, 1978.

[DoD 1985] Department of Defense, *Military standard 2167a: Defense systems software development,* 1985.

[Dorfman 1997] Dorfman, M., "Requirements Engineering", *Software Requirements Engineering* by Thayer. R. and Dorfman, M., IEEE Computer Society, 1997.

[ESPITI 1995] European Software Process Improvement Training Initiative, *User Survey Report*, 1995.

[Fairley 1996] Fairley, R. and Thayer, R., "The Concept of Operations: The Bridge from Operational Requirements to Technical Specifications", *Software Engineering*, 1996 pp. 44-54.

[Gause 1989] Gause, D. and Weinberg, G., *Exploring Requirements: Quality Before Design,* Dorset House Publishing, New York, First edition, 1989.

[Goguen 1993] Goguen, J., and Linde, C., "Techniques for Requirements Elicitation", *International Symposium on Requirements Engineering*, 1993.

[Gomma 1981] Gomma, H. and Scott, D., "Prototyping as a tool in the specification of user requirements", *In Fifth Int. Conf. on Software Engineering,* 1981.

[Gotel 1994] Gotel, O. and Finkelstein, A., "An analysis of the requirements traceability problem", *In Proceedings of the First IEEE International Conference on Requirements Engineering (ICRE94),* 1994.

[Greenspan 1982] Greenspan, S., Mylopoulos, J., and Borgida, A., "Capturing more world knowledge in the requirements specification", *Proceedings of the 6th International Conference on Software Engineering (ICSE6),* Tokyo, Japan, 1982.

[Greenspan 1994] Greenspan, S., Mylopoulos, J., and Borgida, A., "On formal requirements modeling languages: RML revisited", *In Proceedings of IEEE International Conference on Software Engineering (ICSE16),* 1994.

[Hanna 1995] Hanna, M., "Farewell to Waterfall", *Software,* May 1995.

[Herlea 1999] Herlea, D., Jonker, C., Treur, J., and Wijngaards, N., "A Formal Knowledge Level Process Model of Requirements Engineering, Multiple approaches to intelligent systems", *Proceedings of the 12th International Conference on Industrial and Engineering Applications of AI and Expert Systems*, 1999.

[Herlea 2000] Herlea, D., "Challenges in Requirements Engineering", *Computer Science Technical Report*, University of Calgary, 2000.

[Heymans 1998] Heymans, P. and Dubois, E., "Scenario-Based Techniques for Supporting the Elaboration and Validation of Formal Requirements", *Requirements Engineering,* vol. 3, 1998.

[Holt 1997] Holt, J., "Current Practices in Software Engineering: A Survey", *Computing and Control Engineering,* 1997.

[Hull 2002] Hull, E., Jackson, K., and Dick, J., *Requirements Engineering*, Springer, 2002.

[IEEE 1987] IEEE, *Software Engineering Standards,* IEEE Press, 1987.

[IEEE 1990] IEEE, *Standard Glossary of Software Engineering Terminology,* IEEE Press, 1990.

[IEEE 1993] IEEE, *Recommended Practice for Software Requirements Specifications,* IEEE, 1993.

[IEEE 1995] IEEE P1233/D3, *Guide for Developing System Requirements Specification*, December, 1995.

[IEEE 1998] IEEE 1362, *IEEE Guide for Information Technology – System Definition – Concept of Operations (ConOps) Document*, 1998.

[Jackson 1995] Jackson, M., *Software Requirements & Specifications,* Addison Wesley, 1995.

[Jacobson 1992] Jacobson, I., Christerson, M., Jonsson, P., and O Vergaard, G., *Object-Oriented Software Engineering,* Addison Wesley, 1992.

[Jacobson 1999] Jacobson, I., Booch, G., and Rumbaugh, J., *The Unified Software Development Process,* Addison Wesley, 1999.

[Jalote 1999] Jalote, P., *An Integrated Approach to Software Engineering,* Narosa Publications, 1999.

[Jarke 1994] Jarke, M. and Pohl, K, "Requirements Engineering in 2001: (Virtually) Managing a Changing Reality", *Software Engineering,* November 1994.

[Kasser 1998] Kasser, J., and Williams, V., "What Do You Mean You Can't Tell Me If My Project Is in Trouble?", *First European Conference on Software Metrics (FESMA 98)*, 1998.

[Kotonya 1996] Kotonya, G. and Sommerville, I., "Requirements Engineering with Viewpoints", *Software Engineering,* Vol. 11, Jan 1996.

[Kotonya 1998] Kotonya, G. and Sommerville, I., *Requirements Engineering: Processes and Techniques,* John Wiley & Sons, 1998.

[Krasner 1989] Krasner, H., "Requirements Dynamics in Large Software Projects: A Perspective on New Directions in the Software Engineering Process", *11th World Computer Congress (IFIP89),* Elsevier, New York, 1989.

[Kyng 1995] Kyng, M., "Creating Contexts for Design", *in Scenario Based Design* by J.M. Carroll, New York, Wiley, 1995.

[Lauesen 2002] Lauesen, S., *Software Requirements: Styles and Techniques,* Addison Wesley, 2002.

[Lawrence 1996] Lawrence, B., "Unresolved Ambiguity", *American Programmer* 9(5), 1996.

[Leffingwell 2000] Leffingwell, D., and Widrig, D., *Managing Software Requirements,* Addison-Wesley, 2000.

[Leite 1991] Leite, J. and Freeman, P., "Requirements Validation Through Viewpoints Resolution", *IEEE Transactions on Software Engineering,* 1991.

[Leite 1998] Leite, J., "Scenario Evolution", *Dagstuhl-Seminar-Report,* Alemanha, 1998.

[Leite 2001] Leite, J., "Extreme Requirements", *Jornadas de Ingeniería de Requisitos Aplicadas,* Sevilha, June, 2001.

[Lerch 1997] Lerch, F., Ballou, D., and Harter, D., "Using simulation-based experiments for software requirements engineering", *Annals of Software Engineering, special issue on Software Requirements Engineering,* 1997.

[Levene 1982] Levene, A. and Mullery, G., "An investigation of requirement specification languages: Theory and practice", *IEEE Computer,* 1982.

[Lugi 1993] Luqi, "How to use prototyping for requirements engineering", *In Proceedings of the IEEE International Symposium on Requirements Engineering*, 1993.

[Macaulay 1996] Macaulay, L., *Requirements Engineering,* Springer, 1996.

[Maiden 1998] Maiden, N., Minocha, S., Manning, K., and Ryan, M., "CREWS-SAVRE: Systematic scenario generation and use", *In Proceedings of the IEEE Third International Conference on Requirements Engineering (ICRE98),* 1998.

[Maude 1991] Maude, T., *Rapid Prototyping: The Management of Software Risk,* Pitman, 1991.

[McDermid 1993] McDermid, J. and Rook, P., "Software Development Process Models", *Software Engineer's Reference Book,* CRC Press, 1993.

[McDermid 1994] McDermid, J., "Requirements analysis: Orthodoxy, fundamentalism and heresy", in *Requirements Engineering Social and Technical Issues* by Jirotka, M. and Goguen, J., 1994.

[McGraw 1997] McGraw, K., and Harbison, K., *User-Centered Requirements: The Scenario-Based Engineering Process*, Lawrence Erlbaum Associates, 1997.

[Moore 1991] Moore, G., *Crossing the Chasm: Marketing and Selling High-Tech Business Goals*, Addison-Wesley, 1991.

[Mylopoulos 1992] Mylopoulos, J., Chung, L., Yu, E., and Nixon, B., "Representing and Using Non-functional Requirements: A Process-Oriented Approach", *IEEE Trans. On Software Eng*, 18(6), pp: 483-497, June 1992.

[Neill 2003] Neill, C. and Laplante, P., "Requirements Engineering: The State of the Practice", *IEEE Software,* 2003.

[Nubeiseh 2000] Nuseibeh, B. and Easterbrook, S., "Requirements Engineering: a roadmap", ACM Press, 2000.

[Potts 1991] Potts, C., "Seven (plus or minus two) challenges for requirements research", *In 6th Int. Workshop on Software Specification and Design,* IEEE Computer Society Press, October 1991.

[Potts 1994] Potts, C., Takahashi, K., and Anton, A., "Inquiry-based Requirements Analysis", *IEEE Software*, 1994.

[Pressman 2001] Pressman, R., *Software Engineering: A Practitioner's Approach,* McGraw Hill, 2001.

[Rockstrom 1982] Rockstrom, A. and Saracco, R., "SDL-CCITT specification and description language", *IEEE Transactions on Communications,* 1982.

[Rolland 1994] Rolland, C., "Modeling the evolution of artifacts", *In Proceedings of First International Conference on Requirements Engineering (ICRE94),* 1994.

[Royce 1970] Royce, W., "Managing the Development of Large Software Systems: Concepts and Techniques", *Proceedings of the Western Electronic Show and Convention (WesCon),* Los Angeles, August 1970, pp. 1-9 (Reprinted in the Proceedings of 9th International Conference on Software Engineering, March 1987, Monterey CA, pp. 328 . 338.)

[Salo 1997] Salo, A., "On the Measurement of Preferences in the Analytic Hierarchy Process", *Journal Of Multi-Criteria Decision Analysis*, 1997.

[Sawyer 1997] Sawyer, P. and Sommerville, I., "Viewpoints: principles, problems and a practical approach to requirements engineering", *Annals of Software Engineering, special issue on software requirements engineering,* 1997.

[Schach 1996] Schach, S., *Classical and Object-Oriented Software Engineering,* McGraw-Hill, 1996.

[SEI-Risk 1996] SEI, "Software Risk Taxonomy", *Technical Report CMU/SEI-96-TR-012,* 1996.

[Siddiqi 1997] Siddiqi, J., Morrey, I., Roast, C., and Ozcan, M., "Towards quality requirements via animated formal specifications", *Annals of Software Engineering,* 1997.

[Sidky 2002] Sidky, A., Sud, R., Bhatia, S., and Arthur, J., "Problem Identification and Decomposition within the Requirements Generation Process", *Technical Report,* Virginia Tech, http://eprints.cs.vt.edu:8000/archive/00000645/, July 2002.

[Sommerville 1996] Sommerville, I., *Software Engineering*, Addison Wesley Publishing, 1996.

[Sommerville 1998] Sommerville, I., Sawyer, P., and Viller, S., "Viewpoints for requirements elicitation: A practical approach", *In Proceedings of the IEEE 3rd International Conference on Requirements Engineering,* 1998.

[Standish 1995] Standish Group, *CHAOS Chronicles*, 1995.

[Sud 2003] Sud, R., "A Synergistic Approach to Software Requirements Generation: The Synergistic Requirements Generation Model (SRGM) and An Interactive Tool for Modeling SRGM (itSRGM)", *Thesis,* Virginia Tech, 2003.

[Sutcliffe 1998] Sutcliffe, A. and Minocha, S., "Scenario-based analysis on non-functional requirements", *In Proceedings of the Fourth International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ'98)*, 1998.

[Sutcliffe 2002] Sutcliffe, A., *User-Centered Requirements Engineering,* Springer, 2002.

[Systemsguild 2002] www.systemsguild.com, 2002.

[Tavolato 1984] Tavolato, P., and Vincena, K., *A,* "Prototyping Methodology and Its Tool", *In Approaches to Prototyping* by Budde, R., Springer-Verlag, 1984.

[Telelogic 2002] www.telelogic.com, 2002.

[Thayer 1997] Thayer, R., *Software Requirements Engineering*, IEEE Computer Society, 1997.

[Volere 2004] http://www.volere.co.uk/tools.htm, 2004.

[Wallace 1997] Wallace, D., "Verifying and Validating Software Requirements Specifications", *Software Requirements Engineering* by Thayer, R. and Dorfman, M., 1997.

[Weidenhaupt 1998] Weidenhaupt, K., Paul, K., Jarke, M., and Haumer, Pl, "Scenarios in system development: Current practice", *IEEE Software*, March/April 1998.

[Wiegers 1996] Wiegers, K., *Creating a Software Engineering Culture,* Dorset House Publishing, 1996.

[Wiegers 2001] Wiegers K., *Software Requirements,* Microsoft Press, 2001.

[Yeh 1984] Yeh, R., Zave, P., Conn, A., and Cole, G., *Software requirements: New directions and perspectives, Handbook of Software Engineering*, 1984.

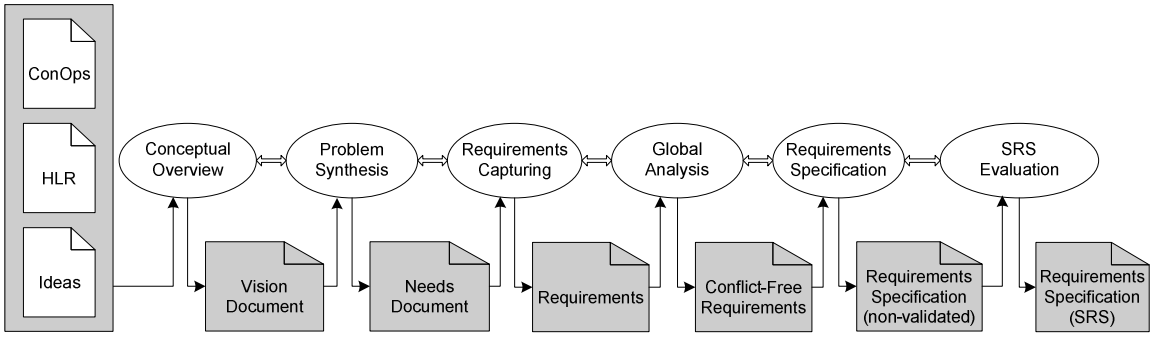
[Yphise 2002] Yphise, "Requirements Management Tools", *Software Assessment Report,* Technology Transfer, 2002.

[Zave 1991] Zave, P., “An insider's evaluation of PAISLey”, *IEEE Transactions on Software Engineering,* 1991.

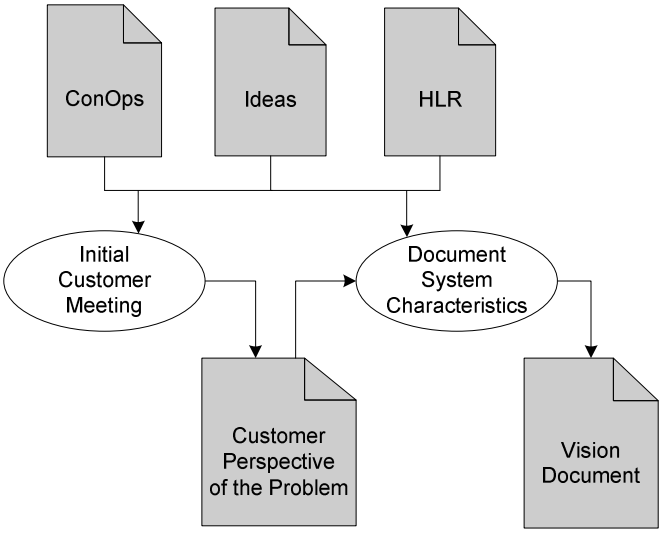
[Zultner 1992] Zultner, R., “Quality Function Deployment for Software: Satisfying Customers”, *American Programmer,* 1992.

# APPENDIX A – THE REFINED RGM MODEL

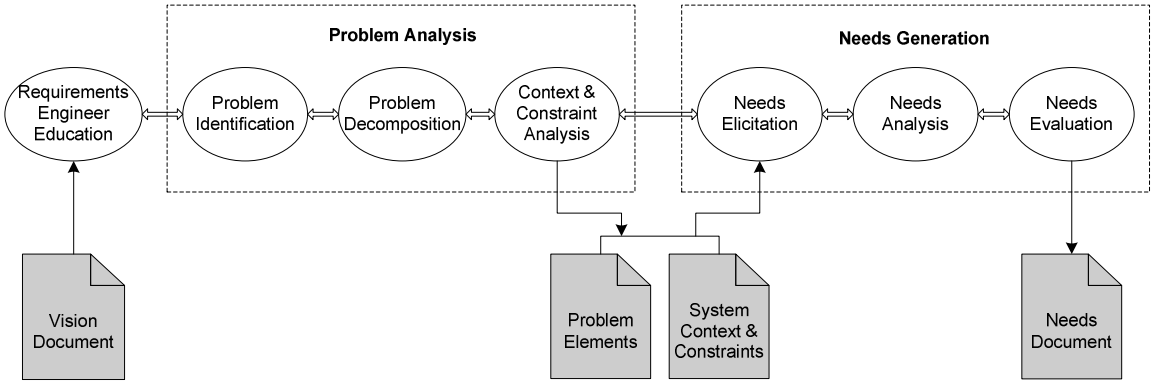
## A. Overview of the Refined RGM Model



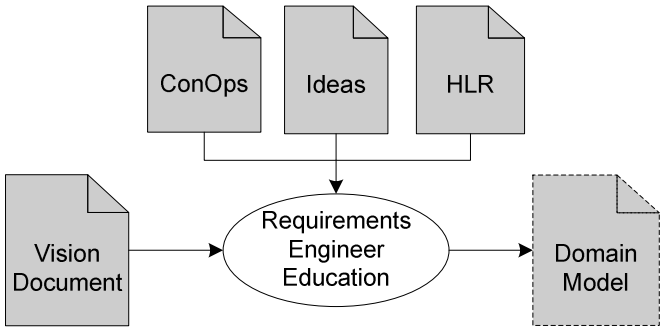

## A.1. Conceptual Overview



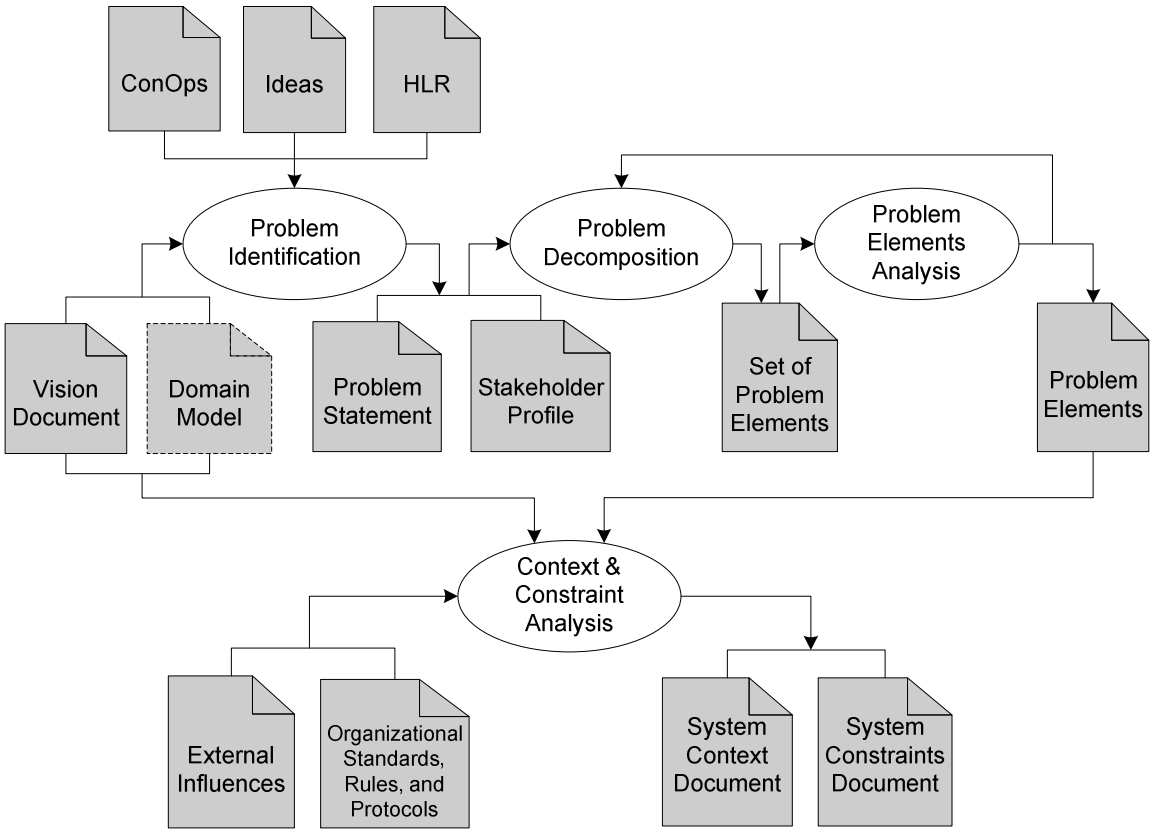
## A.2. Problem Synthesis



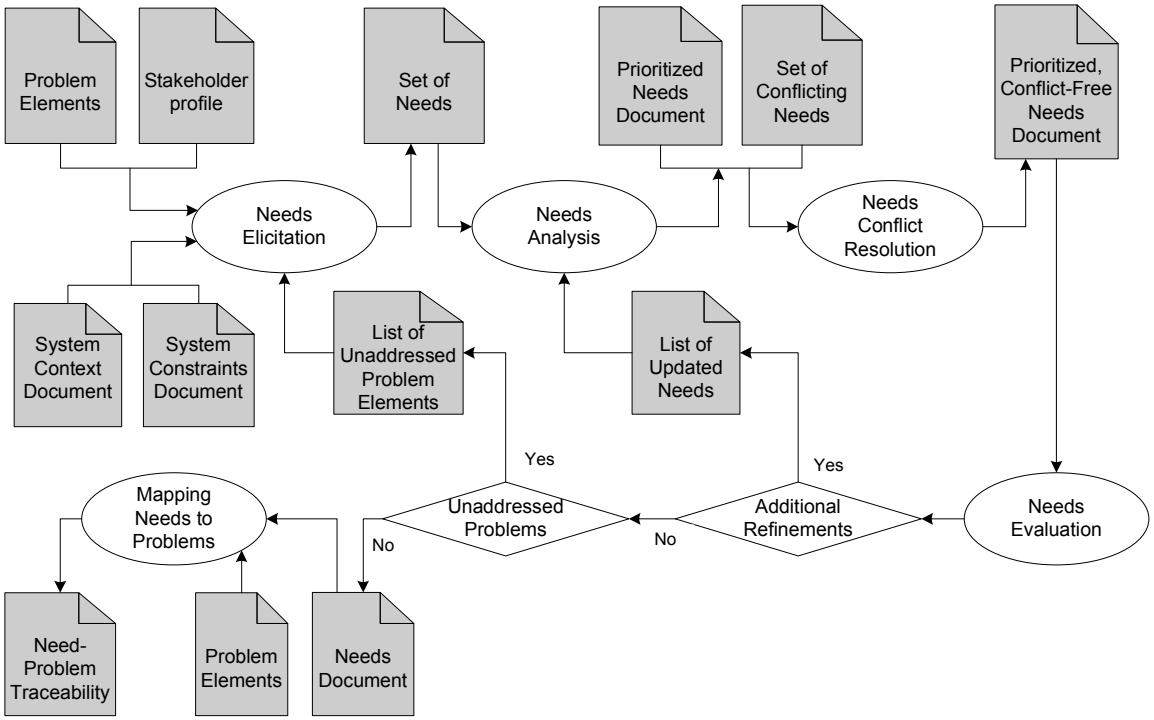

### *A.2.1. Requirements Engineer Education*



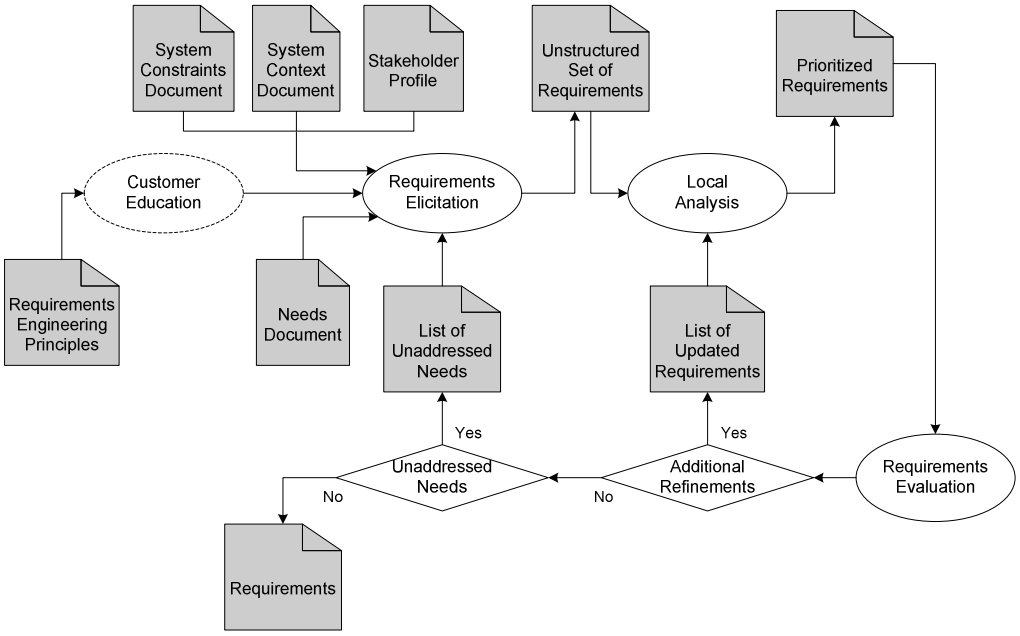
### *A.2.2. Problem Analysis*



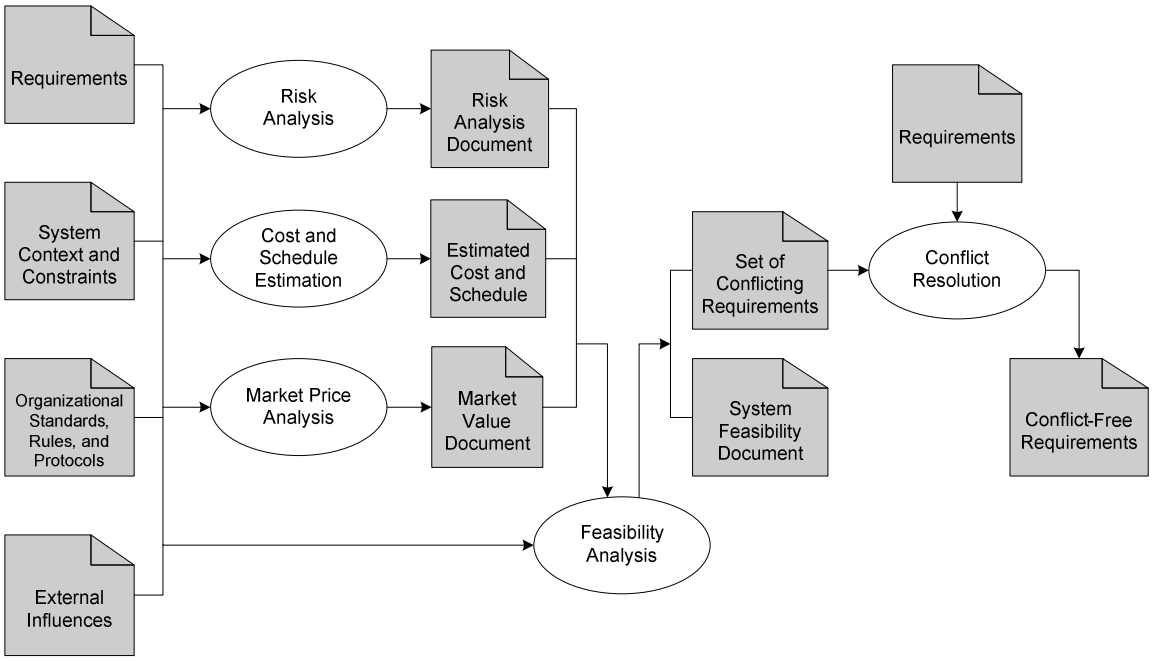
### *A.2.3. Needs Generation*



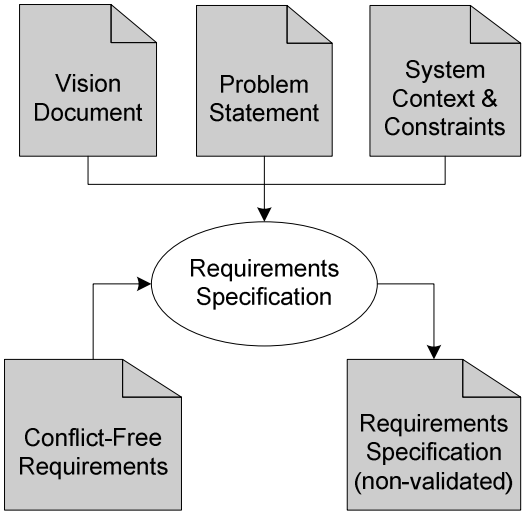
## A.3. Requirements Capturing



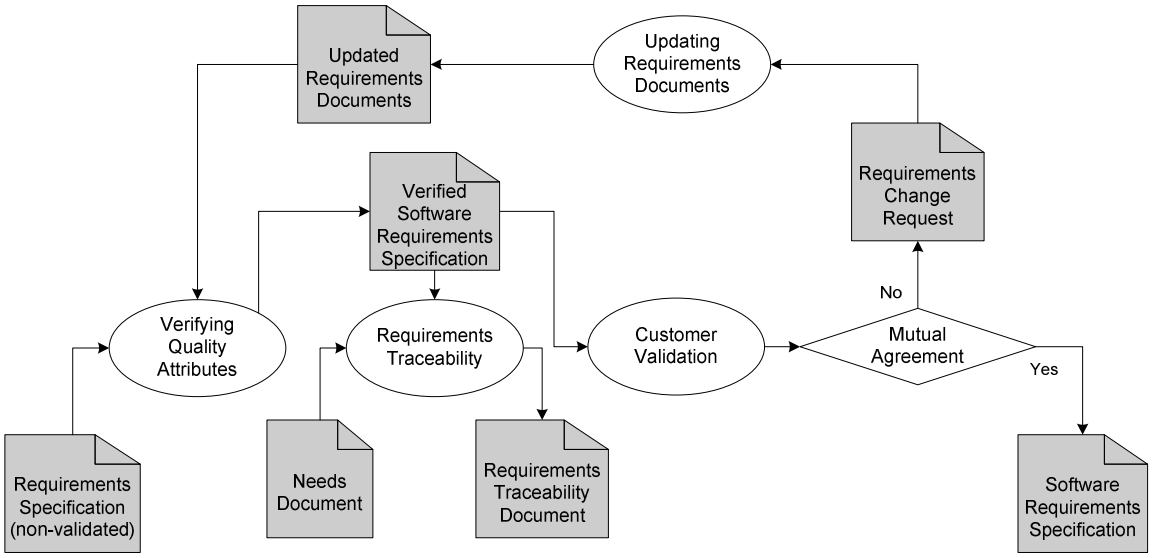
## A.4. Global Analysis



## A.5. Requirements Specification




## A.6. SRS Evaluation



# APPENDIX B. TEMPLATES FOR THE REQUIREMENTS DOCUMENTS

## B.1. Conceptual Overview Documents

### *B.1.1. High Level Requirements*

1. Introduction
    1.1. System Purpose
    1.2. System Scope
    1.3. Definitions and Abbreviations
    1.4. References
2. General System Description
    2.1. Current System
        2.1.1. System Overview and Objectives
        2.1.2. Current Problems and Business Needs
        2.1.3. User Characteristics
    2.2. Proposed System
        2.2.1. System Context
        2.2.2. System Modes and States
        2.2.3. Major System Capabilities
        2.2.4. Major System Constraints

### *B.1.2. ConOps*

1. System Overview
2. Referenced documents
3. Current system or situation
    3.1. Description of the Current System or Situation
    3.2. Operational Policies and Constraints
    3.3. User Classes and Other Involved Personnel
4. Justification for and Nature of Changes
    4.1. Justification of Changes
    4.2. Description of Desired Changes
    4.3. Priorities Among Changes
5. Concepts for the Proposed System
    5.1. Objectives and Scope
    5.2. Description of the Proposed System
6. Operational Scenarios
7. Analysis of the Proposed System
    7.1. Summary of Improvements
    7.2. Disadvantages and Limitations
    7.3. Alternatives and Trade-offs Considered
8. Glossary

### *B.1.3. Customer Perspective of the Problem*

The Customer Perspective of the Problem has the same content as the Vision document (Appendix B.1.4). However, the format follows a question-answer pattern. The information in the document is guided by the following list of sample questions:

- What are the problems faced in the system?
- What are the possible solutions to overcome them?
- What are the short term benefits of this?
- How do you vision the benefits of the new system in the long term?
- What are the losses of the proposed system?

### *B.1.4. Vision Document*

1. Introduction
    1.1. Document Overview
    1.2. Terminology
2. Business Need
    2.1. Existing System Description
    2.2. Problems in the Current System/Situation
    2.3. Organizational Constraints
    2.4. User Classes
3. Rationale
    3.1. Business Opportunities
    3.2. Reasons and Convictions for Change
4. System Concept
    4.1. Proposed System Overview
    4.2. High Level Requirements
5. Business Impact
    5.1. Proposed System Impact on Organization
    5.2. Changes in Current Organizational Setup
    5.3. Risks Involved with System Deployment

## B.2. Problem Synthesis Documents

### *B.2.1. Domain Model*

- Application Domain Overview
    - Background
    - Domain Model [21]
- Business Entities/Components
    - Description
    - Major Functionalities
    - Relationship of Entities

[21] Dataflow Diagrams (Appendix B.3.3.1) can be used to represent Domain Models [Lauesen 2002].

### *B.2.2. Problem Statement*

The following information is recorded, for each high-level problem:

| Information | Description |
|---|---|
| The Problem of | *Describe the problem* |
| Affects | *List of stakeholders affected by the problem* |
| The result of which | *Impact of this problem business activities* |
| Benefits of resolving | *Indicate the proposed solution and list a few key benefits* |

*Table B.1 – Problem Statement Description*

### *B.2.3. Stakeholder Profile*

For each stakeholder, the following information is documented:

| No. | Stakeholder Name | Position | Department | Contact Information | Class of knowledge | Expertise level |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |

*Table B.2 –Stakeholder Information*

### *B.2.4. (Set of) Problem Elements*

The Set of Problem Elements and the Problem Elements documents have similar content and format. The only difference is that the Problem Elements document includes a complete set of problem elements that are ranked.

**General Template of the Problem Elements:**

- Problem 1
    - Problem 1.1.
        - Problem 1.1.1.
        - Problem 1.1.2.
        - …
    - Problem 1.2.
        - Problem 1.2.1.
        - Problem 1.2.2.
        - …
- Problem 2
    - Problem 2.1.

    …

**For each problem element, the following information is recorded:**

- Problem Element #
- Description
- Priority Level
- Affected Stakeholders
- Results of the Problem
- Benefits of Resolving.

### *B.2.5. System Context Document*

1. Introduction
2. Context Diagram (Figure B.2)
3. Context Description
    3.1. System
    3.2. External Entities
        3.2.1. Related Data Flow

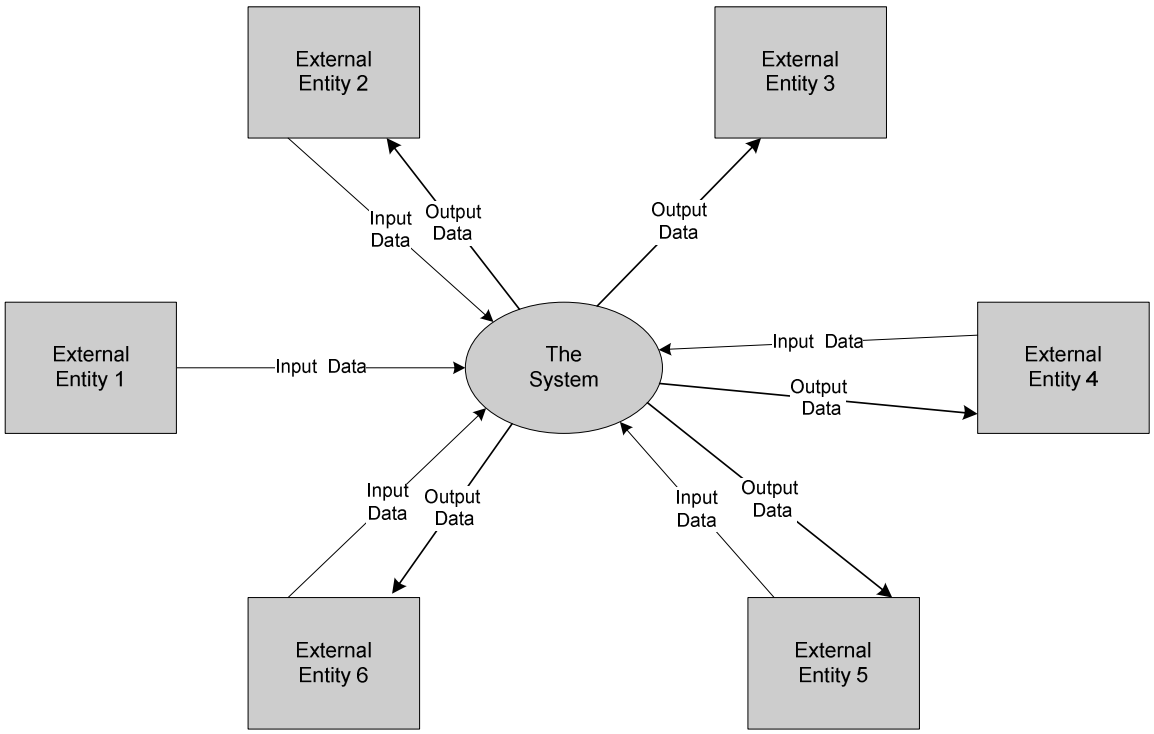



*Figure B.1 – Context Diagram*

Notations Used:

- System – the system/product
- External Entity – outside system or actor
- Input/Output Data – data flow between the system and the external entities.

### *B.2.6. System Constraints Document*

| Category | Constraint | Rationale |
|---|---|---|
| **Economic** | Constraint 1 | Rationale |
| | Constraint 2 | Rationale |
| | … | … |
| **Political** | | |
| | | |
| **Technical** | | |
| | | |
| **System** | | |
| | | |
| **Environmental** | | |
| | | |
| … | | |

*Table B.3 – System Constraints*

### *B.2.7. Set of Needs*

The general template for this document is as follows:

- **Problem 1**
    - Need 1.1
    - Need 1.2
    - **Problem 1.1**
        - Need 1.1.1
        - Need 1.1.2
        - …
    - **Problem 1.2**
        - …
- **Problem 2**
    - Need 2.1
    - …
    - **Problem 2.1**
        - Need 2.1.1
        - Need 2.1.2
        - …

### *B.2.8. Structured Set of Needs document*

The following is the general structure of the needs:

- Feature 1
    - Need 1.1
    - Need 1.2
    - …
- Feature 2
    - Need 2.1
    - Need 2.2
    - …
- User Scenarios (provides supplementary information)

For each need, the following information is documented:

- Need #
- Description
- Rationale (refers to related Problem Element)
- Source/author.

### *B.2.9. Prioritized Needs Document*

The general structure of the Prioritized Needs Document is similar to the Structured Set of Needs (Appendix B.2.8); the only additional feature is the needs are ranked based on their importance to the stakeholders.

The following information is recorded for each need:

- Need #
- Priority level
- Description
- Rationale (refers to related Problem Element)
- Source/author.

### *B.2.10. Set of Conflicting Needs*

For each conflict, the following information is documented:

- List of Conflicting Needs:
    - Need 1
    - Need 2
    - …
- Reason for the Conflict
- Possible Solutions.

### *B.2.11. Prioritized, Conflict-Free Needs Document*

The content and format of the Prioritized, Conflict-Free Needs Document is the same as the Prioritized Needs document (Appendix B.2.9), the difference being that the needs are conflict-free.

### *B.2.12. List of Updated Needs*

For each updated need, the following information is documented:

- Need #
- Original Need
- Modified Need
- Reason for the Change
- Author of the Change
- Date of Modification (optional)
- Impact of Change (optional).

### *B.2.13. Needs Document*

The content and format of the Needs document is the same as in Prioritized Needs document (Appendix B.2.9), which consists of structured and ranked needs. However, the Needs document includes the complete set of needs, which is refined and validated by the stakeholders.

### *B.2.14. Need-Problem Traceability*

| Problem Elements / Needs | … | P2.11 | P2.12 | P2.13 | P2.14 | P3 | P3.1 | P3.2 | P3.4 | … |
|---|---|---|---|---|---|---|---|---|---|---|
| … | | | | | | | | | | |
| N1 | | | | | | | | | ⬏ | |
| N1.1 | | | ⬏ | | | | | | | |
| N1.2 | | | | | | ⬏ | | | | |
| N1.3 | | | | | | | | | | |
| N1.4 | | ⬏ | | | | | | | | |
| N2 | | | | | | | | | ⬏ | |
| N2.1 | | | | ⬏ | | | | | | |
| N2.2 | | | | | | | | | | |
| … | | | | | | | | | | |

*Figure B.2 – Need-Problem Traceability Matrix*

## B.3. Requirements Capturing Documents

### *B.3.1. Unstructured Set of Requirements*

The requirements are organized as follows:

- **Need 1**
    - Requirement 1.1
    - Requirement 1.2
    - **Need 1.1**
        - Requirement 1.1.1
        - Requirement 1.1.2
        - …
    - **Need 1.2**
        - …
- **Need 2**
    - Requirement 2.1
    - …
    - **Need 2.1**
        - Requirement 2.1.1
        - Requirement 2.1.2
        - …
- Scenarios (provides supplementary information)

## *B.3.2. Models*

### B.3.2.1. Dataflow Diagram (DFD)

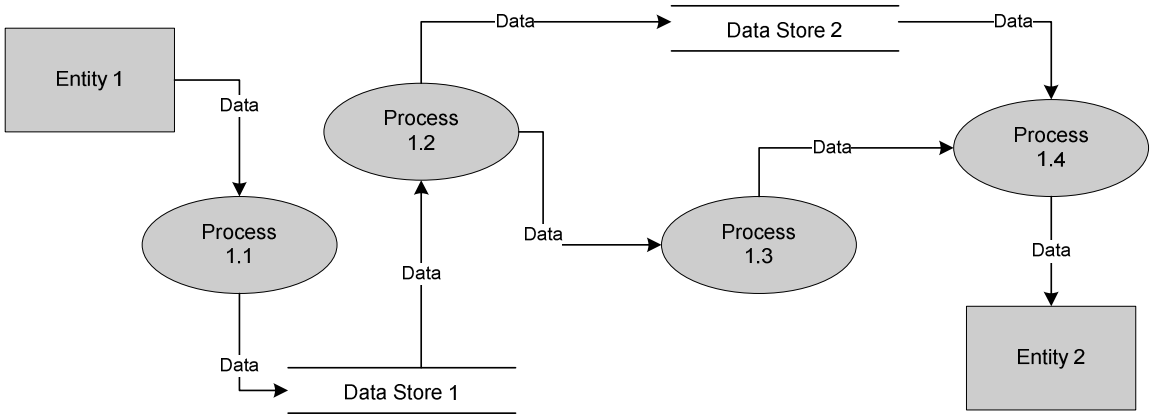



*Figure B.3 – Dataflow Diagram*

Notations used:

- Squares represent entities, which are sources or destinations of data.
- Rounded rectangles/ovals represent processes that convert incoming data to information.
- Arrows represents the data flows, which can be either electronic data or physical items.
- Open-ended rectangles represent data stores such as databases or XML files.

### B.3.2.2. State Transition Model

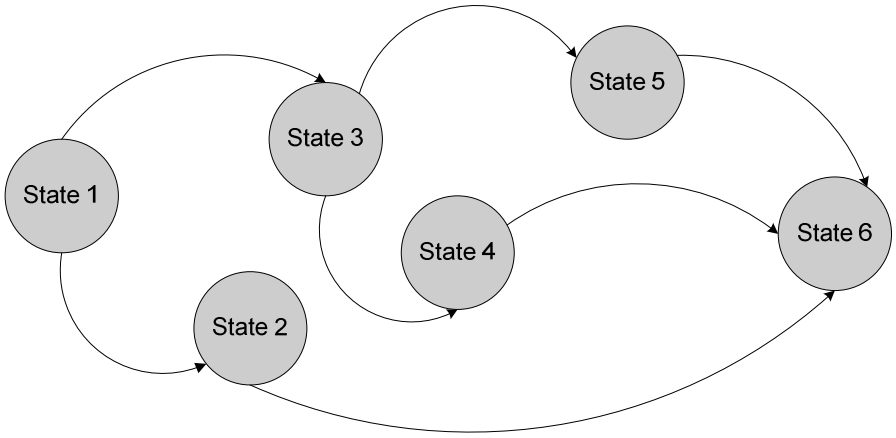



*Figure B.4 – State Transition Model*

Notations used:

- Circle denotes the state.
- Directed arc connecting the two states denotes the transition to the next state**.**

### B.3.2.3. Data Model (Entity-Relationship Model)

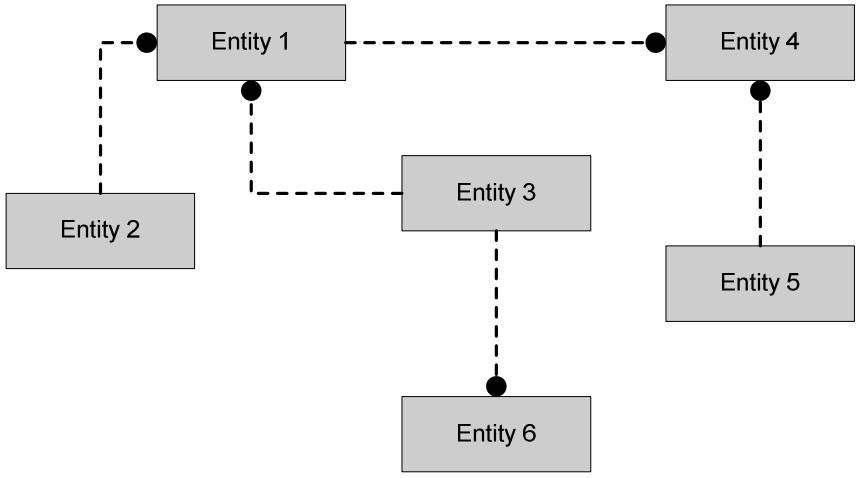



*Figure B.5 – Entity-Relationship Diagram*

- Entity – object or concept.
- Relationship ( -----• ) – represents two entities sharing information in the database structure.
- Cardinality – specifies how many instances of an entity relate to one instance of another entity.
    - -----• 1-N (One-to-many relationship)
    - •----- N-1 (Many-to-one relationship)

### B.3.2.4. Decision Tree

A decision tree is a diagram that represents the possible consequences of a series of decisions in a particular situation.

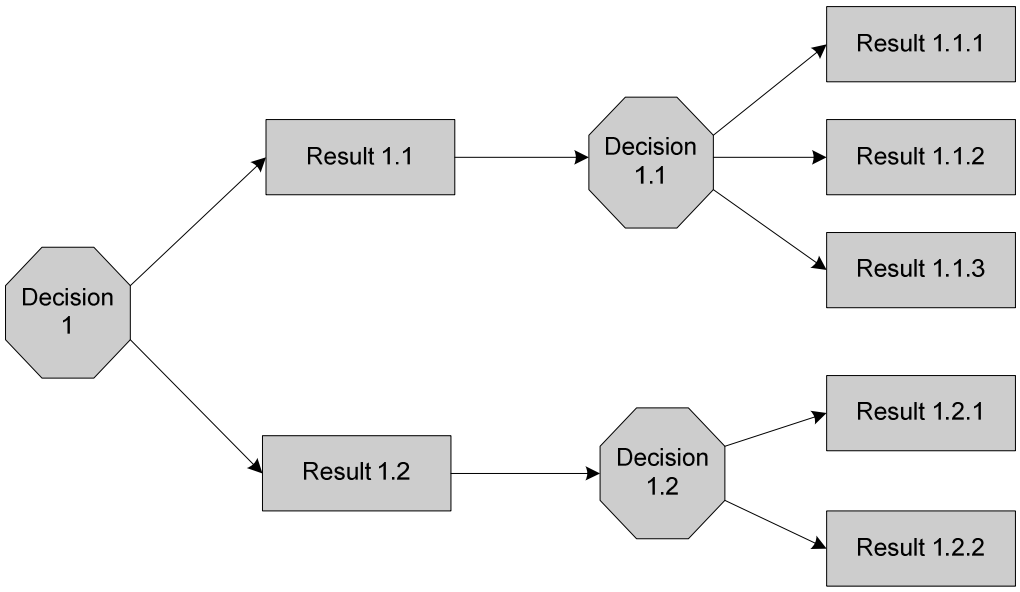



*Figure B.6 – Decision Tree*

### B.3.2.5. Data dictionary

| |
|---|
| Name (of the Data Item): |
| Description/Purpose: |
| Attributes (Properties or Characteristics of the Data Item): |
| Related Data Items: |

*Table B.4 – Data Dictionary*

### *B.3.3. Structured Set of Requirements*

For each requirement, the following information is documented:

| |
|---|
| Requirement #: |
| Description: |
| Rationale (*references related user need*): |
| Source (*originator*): |
| Dependencies: |
| Supporting materials (*references related scenarios and/or models*): |
| History (*creation and modification dates*): |

*Table B.5 –Requirement Template in the Structured Set of Requirements*

The requirements are categorized as functional and non-functional.

#### B.3.3.1. Functional Requirements

**Organized by Functional Hierarchy**

- Functionality 1
    - Functional Requirement 1.1
    - Functional Requirement 1.2
    - …
- Functionality 2
    - Functional Requirement 2.1
    - …
- …

**Organized by Mode**

- Mode 1
    - Functional Requirement 1.1
    - Functional Requirement 1.2
    - …
- Mode 2
    - Functional Requirement 2.1
    - …
- …

**Organized by User Class**

- User Class 1
    - Functional Requirement 1.1
    - Functional Requirement 1.2
    - …
- User Class 2
    - Functional Requirement 2.1
    - …
- …

**Organized by Stimulus**

- Stimulus 1
    - Functional Requirement 1.1
    - Functional Requirement 1.2
    - …
- Stimulus 2
    - Functional Requirement 2.1
    - …
- …

#### B.3.3.2. Non-functional Requirements

- Safety
- Security
- Performance
- Reliability
- Usability
- Maintainability
- Portability
- Interoperability
- …

### *B.3.4. Prioritized Requirements*

The format of the Prioritized Requirements document is similar to the Structured Set of Requirements (Appendix B.3.3) with the difference being that the requirements are now ranked and have attributes such as risk factors, effort, and user importance.

For each requirement, the following information is recorded:

| | |
|---|---|
| Requirement #: | Priority Level |
| Description: | |
| Rationale (*references related user need*): | |
| Risks factors involved: | Effort: |
| Source (*originator*): | |
| Dependencies: | |
| Supporting materials (*references related scenarios and/or models*): | |
| History (*creation and modification dates*): | |

*Table B.6 – Requirement Template in the Prioritized Set of Requirements*

### *B.3.5. List of Updated Requirements*

For each updated requirement, the following information is documented:

- Requirement #
- Original Requirement
- Modified Requirement
- Reason for Change
- Author of the Change
- Date of modification
- Impact of Change.

### *B.3.6. Requirements*

The format of the Requirements document is the same as the Prioritized Requirements document (Appendix B.3.4) with the only difference being that the requirements are complete, verified, and validated.

## B.4. Global Analysis Documents

The format for the documents generated in the Global Analysis phase varies for different organizations. Hence, only the general information present in these documents is included in this Appendix.

### *B.4.1. Risk Analysis Document*

- Evaluation of requirements for the various risk factors
- High risk requirements
- Recommendations for risk mitigation

### *B.4.2. Estimated Cost and Schedule*

- Cost estimation calculations
- Cost intensive requirements
- Schedule estimation calculations
- Time intensive requirements

### *B.4.3. Market Survey Document*

- Number of customers and competitors in the market
- Prices of similar/competitor products
- Intensity of demand
- Quality of products in the market
- Important/desirable product features

### *B.4.4. Market Value Document*

Statistics involved in obtaining the price estimate.

- For each product feature:
    - Relevant market information
        - Competitor's price
        - User demand
        - Reasonable price
- Calculations for price estimation
- Features that do not add significant value to the product

### *B.4.5. System Feasibility Document*

- Feasibility Evaluations
    - Operational feasibility
    - Technical feasibility
    - Schedule feasibility
    - Economic feasibility
- Overall profitability of the product
- Decision whether the project is feasible or not
    - Reasoning for the decision

### *B.4.6. Set of Conflicting Requirements*

For each conflict, the following information is documented:

- List of Conflicting Requirements:
    - Requirement 1
    - Requirement 2
    - …
- Reason for the Conflict
- Possible Solutions.

### *B.4.7. Conflict-Free Requirements*

The content and format of the Conflict-Free Requirements document is the same as the Requirements document (Appendix B.3.6) with the difference being that the requirements are conflict-free.

## B.5. Requirements Specification Documents

### *B.5.1. (Non-Validated) Requirements Specification*

1. Introduction
    1.1. Purpose
    1.2. Document Conventions
    1.3. Definitions and Abbreviations
    1.4. Intended Audience
    1.5. Product Scope
    1.6. References
2. Overall Description
    2.1. Product Perspective
    2.2. Product Functions
    2.3. User Classes and Characteristics
    2.4. Operating Environment
    2.5. Design and Implementation Constraints
    2.6. Assumptions and Dependencies
3. External Interface Requirements
    3.1. User Interfaces
    3.2. Hardware Interfaces
    3.3. Software Interfaces
4. System Features
    4.1. System Feature
    4.2. Description and Priority

4.3. Functional Requirements (organized according to the templates provided in Appendix B.3.3)

5. Non-functional Requirements

5.1. Performance Requirements

5.2. Safety Requirements

5.3. Security Requirements

5.4. Usability

5.5. Reliability

5.6. Maintainability

5.7. Portability

6. Other Requirements

7. Appendix

7.1. Glossary

7.2. To-Be-Determined List

7.3. Scenarios and Models

## B.6. SRS Evaluation Documents

### *B.6.1. Verified Requirements Specification*

The content and format of the document is similar to the Requirements Specification document (Appendix B.5.1); the difference is that the Verified Requirements Specification document includes a section which lists the requirements that do not adhere to the quality characteristics.

| Functionalities/ Requirements | Problem found | Description of the Problem | Recommendations |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

*Table B.7 – Requirements Verification Report*

### *B.6.2. Requirements Traceability Document*

| Requirements or needs / Requirements | … | R2.11 | R2.12 | R2.13 | R2.14 | N3 | N3.1 | N3.2 | N3.4 | … |
|---|---|---|---|---|---|---|---|---|---|---|
| … | | | | | | | | | | |
| R1 | | | | | | | | | ⮤ | |
| R1.1 | | | ⮤ | | | | | | | |
| R1.2 | | | | | | ⮤ | | | | |
| R1.3 | | | | | | | | | | |
| R1.4 | | ⮤ | | | | | | | | |
| R2 | | | | | | | | | ⮤ | |
| R2.1 | | | | ⮤ | | | | | | |
| R2.2 | | | | | | | | | | |
| … | | | | | | | | | | |

*Figure B.7 – Requirements Traceability Matrix*

### *B.6.3. Requirements Change Request*

For each change request, the following information is included:

- Change Request #
- Title
- Source (author of the modification)
- Date
- Original Requirement
- Requested Change
- Reason for the Change
- Impact of the Change
- Consequences if Not Approved.

### *B.6.4. Software Requirements Specification*

The final output of the requirements generation process, SRS, has the same format as the Requirements Specification document (Appendix B.5.1) with the only difference being that the SRS is verified for quality attributes and validated by the customers.

# APPENDIX C – TRANSFORMATION OF THE REQUIREMENTS DOCUMENTS